**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

Курсовая РАБОТА

**по дисциплине «Программирование»**

Тема: Работа с изображениями

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Колесниченко М. А. |
| Преподаватель |  | Глазунов С. А. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент: Колесниченко М. А. | | |
| Группа: 3342 | | |
| Тема работы: Работа с изображениями | | |
| Исходные данные:  Вариант 5.16  Программа обязательно должна иметь CLI (опционально дополнительное использование GUI). Программа должна реализовывать весь следующий функционал по обработке png-файла  Общие сведения:   * Формат картинки PNG (рекомендуем использовать библиотеку libpng) без сжатия * файл может не соответствовать формату PNG, т.е. необходимо проверка на PNG формат. Если файл не соответствует формату PNG, то программа должна завершиться с соответствующей ошибкой. * обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями. * все поля стандартных PNG заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).   Программа должна иметь следующие функции по обработке изображений:   * Копирование заданной области. Флаг для выполнения данной операции: `--copy`. Функционал определяется:  1. Координатами левого верхнего угла области-источника. Флаг `--left\_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y 2. Координатами правого нижнего угла области-источника. Флаг `--right\_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y 3. 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  * Заменяет все пиксели одного заданного цвета на другой цвет. Флаг для выполнения данной операции: `--color\_replace`. Функционал определяется:  1. 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) 2. Цвет на который требуется заменить. Флаг `--new\_color` (работает аналогично флагу `--old\_color`)  * Сделать рамку в виде узора. Флаг для выполнения данной операции: `--ornament`. Рамка определяется:  1. Узором. Флаг `--pattern`. Обязательные значения: rectangle и circle, semicircles. Также можно добавить свои узоры (красивый узор можно получить используя фракталы). Подробнее здесь: <https://se.moevm.info/doku.php/courses:programming:cw_spring_ornament> 2. 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) 3. Шириной. Флаг `--thickness`. На вход принимает число больше 0 4. Количеством. Флаг `--count`. На вход принимает число больше 0 5. При необходимости можно добавить дополнительные флаги для необозначенных узоров  * 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:  1. Цветом искомых прямоугольников. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет) 2. Цветом линии для обводки. Флаг `--border\_color` (работает аналогично флагу `--color*`*) 3. Толщиной линии для обводки. Флаг `--thickness`. На вход принимает число больше 0   Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов (например, функции обработки текста в один, функции ввода/вывода в другой). Сборка должна осуществляться при помощи make и Makefile или другой системы сборки | | |
| Разделы пояснительный записки: «Содержание», «Введение», «Структуры», «Функции», «Заключение», «Список использованных источников», «Приложение А. Примеры работы программы», «Приложение Б. Исходный код программы». | | |
| Предполагаемый объем пояснительной записки:  Не менее 20 страниц. | | |
| Дата выдачи задания: 18.03.2024 Дата сдачи реферата: 10.05.2024 Дата защиты реферата: 15.05.2024 | | |
| Студент |  | Колесниченко М. А. |
| Преподаватель |  | Глазунов С. А. |

**Аннотация**

Курсовая работа заключается в создании программы для обработки изображений в формате PNG на языке С. Она представляет собой набор компонентов для обработки файлов формата PNG, включающих функции копирования, замены цветов, орнаментации и наложения рамок вокруг заполненных прямоугольников. Структура проекта включает каталоги src для файлов исходного кода, include для заголовочных файлов, docs для документации, Makefile для сборки проекта и README.md с описанием. Для сборки проекта используется команда make, а для генерации документации — make docs, которая использует Doxygen. После сборки проекта можно выполнять различные операции с файлами PNG, указывая соответствующие параметры командной строки. Программа зависит от библиотек libpng и математической библиотеки math, поэтому перед сборкой убедитесь, что они установлены в системе.

Пример работы программы приведен в приложении А.

Исходный код программы приведен в приложении Б.

**ANNOTATION**

The course work consists in creating a program for processing PNG images in the C language. It is a set of components for processing PNG files, including functions for copying, replacing colors, ornamentation and applying frames around filled rectangles. The project structure includes src directories for source code files, include for header files, doc for documentation, Makefile for building the project and README.md with a description. The make command is used to build the project, and make docs, which uses Doxygen, is used to generate documentation. After building the project, you can perform various operations with PNG files by specifying the appropriate command line options. The program depends on the libpng and math libraries, so make sure they are installed on the system before building.

An example of how the program works is given in Appendix A.

The source code of the program is given in Appendix B.

**содержание**

[Введение 8](#_Toc168242901)

[1. Структуры 9](#_Toc168242902)

[1.1 Структура Png 9](#_Toc168242903)

[1.2 Структура Area 9](#_Toc168242904)

[1.3 Структура Options 10](#_Toc168242905)

[2. Функции 11](#_Toc168242906)

[2.1 Функция main 11](#_Toc168242907)

[2.2 Подготовительные функции 11](#_Toc168242908)

[2.3 Функции для работы с файлами 12](#_Toc168242909)

[2.4 Функции выполнения заданий 12](#_Toc168242910)

[2.5 Функции рисования 14](#_Toc168242911)

[Заключение 15](#_Toc168242912)

[Список использованных источников 16](#_Toc168242913)

[Приложение А 17](#_Toc168242914)

[Приложение Б 19](#_Toc168242915)

Введение

Цель работы: написать программу на языке С, которая считывает изображение и обрабатывает его требуемым пользователем образом. Для этого требется реализовать:

* Загрузку, хранение изображений из файла и запись изображения в файл;
* Ввод аргументов из коммандной строки;
* Функции для рисования на загруженном изображении.

1. Структуры

Чтобы облегчить написание и чтение кода, было принято решение реализовать и использовать 3 основные структуры: Png, Area, Options.

1.1 Структура Png

Структура Png представляет изображение в формате PNG. Она содержит следующие поля:

- `width`: Ширина изображения в пикселях.

- `height`: Высота изображения в пикселях.

- `color\_type`: Тип цвета изображения (например, RGB, оттенки серого).

- `bit\_depth`: Глубина цвета изображения.

- `png\_ptr`: Указатель на структуру libpng для чтения/записи данных PNG.

- `info\_ptr`: Указатель на структуру libpng для хранения информации о PNG.

- `number\_of\_passes`: Количество проходов, необходимых для интерлейсинга.

- `row\_pointers`: Указатель на массив указателей, каждый из которых указывает на строку данных изображения.

Эта структура необходима для представления информации о PNG-изображении в программе, а также для работы с библиотекой libpng для чтения и записи данных в формате PNG.

1.2 Структура Area

Структура Area представляет область, скопированную во время выполнения функции 'copy\_area'. Включает в себя следующие поля:

- `width`: Ширина области в пикселях.

- `height`: Высота области в пикселях.

- `row\_pointers`: Указатель на массив указателей, каждый из которых указывает на строку данных области.

Эта структура используется для хранения информации о скопированной области в процессе выполнения функции 'copy\_area'.

1.3 Структура Options

Эта структура хранит в себе большое количество полей, которые отвечают за флаги, передаваемые пользователем при запуске. Из нее можно получить информацию, был ли передан определенный флаг и его значение.

2. Функции

2.1 Функция main

Функция `main` является главной функцией программы, которая обрабатывает аргументы командной строки и выполняет задачи над изображением. Она принимает аргументы `argc` (количество аргументов командной строки) и `argv` (массив строк, содержащий аргументы командной строки).

Внутри функции происходят следующие шаги:

Инициализация структуры `Options` с значениями по умолчанию и установка имени выходного файла по умолчанию. Затем происходит разбор аргументов командной строки с помощью функции `handle\_arguments`.

После этого инициализируется структура `Png` для хранения информации о входном PNG-файле, который читается с помощью функции `read\_png\_file`.

Далее выполняется обработка задач на основе предоставленных параметров с помощью функции `task\_switcher`.

И, наконец, измененное изображение записывается в выходной PNG-файл с помощью функции `write\_png\_file`.

Функция завершает свою работу, возвращая целочисленный код статуса завершения программы. В данном случае возвращается 0, что обычно означает успешное завершение программы.

2.2 Подготовительные функции

Функция `handle\_arguments` обрабатывает аргументы командной строки, переданные программе, и соответствующим образом заполняет структуру `Options`. Передается количество аргументов командной строки `argc`, массив строк, содержащий аргументы командной строки `argv`, и указатель на структуру `Options`, где будут храниться разобранные аргументы.

Функция `process\_color` преобразует цвет, предоставленный в виде строки, в целочисленный массив, содержащий красную, зеленую и синюю компоненты цвета. Возвращает указатель на этот массив.

Функция `process\_coordinates` преобразует координаты, предоставленные в виде строки, в целочисленный массив, содержащий координаты X и Y. Возвращает указатель на этот массив.

2.3 Функции для работы с файлами

Функция `read\_png\_file` считывает PNG-файл и сохраняет его информацию и данные пикселей в структуре `Png`. Принимает строку `file\_name`, представляющую имя файла/путь к PNG-изображению, и указатель на структуру `Png`, где будет сохранена информация и данные об изображении.

Функция `write\_png\_file` записывает PNG-изображение в файл. Принимает строку `file\_name`, представляющую имя файла/путь, куда будет сохранено PNG-изображение, и указатель на структуру `Png`, содержащую информацию о PNG-изображении.

Обе функции открывают файл, проверяют его на соответствие формату PNG, инициализируют структуры для чтения/записи PNG-данных, устанавливают обработчики ошибок, инициализируют ввод/вывод, записывают информацию об изображении, записывают данные изображения, завершают процесс записи и освобождают ресурсы.

2.4 Функции выполнения заданий

Функция `print\_help` выводит справочное сообщение, объясняющее использование программы и ее опций. Выводит информацию о курсовой работе и ее создателе, синтаксис использования, а также описания доступных опций, включая краткие и длинные формы и их соответствующие объяснения.

Функция `print\_png\_info` выводит информацию о PNG-изображении. Принимает указатель на структуру `Png`, содержащую информацию о PNG-изображении. Выводит различные детали о PNG-изображении, включая его ширину, высоту, тип цвета, глубину цвета и количество проходов. Тип цвета выводится как строковое представление, глубина цвета указывает на количество битов на выборку или на канал в изображении, а количество проходов относится к количеству проходов, необходимых для интерлейсированных PNG-изображений.

Функция `color\_replace` заменяет все пиксели указанного старого цвета на новый цвет. Принимает указатель на структуру `Png`, представляющую изображение, строку `old\_color`, представляющую старый цвет в формате "R,G,B", и строку `new\_color`, представляющую новый цвет в том же формате.

Функция `copy\_area` копирует указанную область из исходного изображения в новую структуру, а затем копирует ее обратно в исходное изображение в другом месте. Принимает указатель на структуру `Png`, представляющую исходное изображение, строки, содержащие координаты левого верхнего угла и правого нижнего угла области для копирования, а также строку с координатами левого верхнего угла местоположения назначения в исходном изображении для скопированной области.

Функция `filled\_rects` находит все заполненные прямоугольники на изображении и рисует вокруг них границы. Принимает указатель на структуру `Png`, представляющую изображение, строки `string\_color` и `string\_border\_color`, представляющие цвет заполненных прямоугольников и цвет границы соответственно, а также строку `thickness`, представляющую толщину границы.

Функция `ornament` рисует узорный узор на заданном изображении. Принимает указатель на структуру `Png`, представляющую изображение, строку `pattern`, указывающую тип узора ("rectangle", "circle", "semicircles"), строки `string\_color` и `thickness`, представляющие цвет узора и толщину соответственно, а также строку `count`, представляющую количество узоров, которые следует нарисовать.

Функция `task\_switcher` обрабатывает переключение задач на основе предоставленных опций. Принимает структуру `Options`, содержащую флаги и значения для различных задач, и указатель на структуру `Png`, представляющую изображение.

2.5 Функции рисования

Функция `draw\_pixel` рисует один пиксель с указанными значениями цвета. Принимает указатель `ptr` на пиксель в изображении и массив `color\_values`, содержащий значения RGB цвета пикселя.

Функция `draw\_border` рисует границу вокруг указанного прямоугольника на изображении. Принимает указатель на структуру `Png`, представляющую изображение, координаты прямоугольника `x1`, `y1`, `x2`, `y2`, массив `border\_color` с значениями RGB цвета границы и строку `thickness`, представляющую толщину границы.

Функция `rectangle\_ornament` рисует узорные прямоугольники на изображении. Принимает указатель на структуру `Png`, представляющую изображение, толщину орнамента `ornament\_thickness`, количество орнаментов `ornament\_count`, массив `color\_values` с значениями RGB цвета орнамента и строку `thickness`, представляющую толщину границы.

Функция `circle\_ornament` рисует круглый узор на изображении. Принимает указатель на структуру `Png`, представляющую изображение, и массив `color\_values` с значениями RGB цвета орнамента.

Функция `semicircles\_ornament` рисует узорные полукруги на изображении. Принимает указатель на структуру `Png`, представляющую изображение, толщину орнамента `ornament\_thickness`, количество орнаментов `ornament\_count`, массив `color\_values` с значениями RGB цвета орнамента и строку `thickness`, представляющую толщину границы.

Разработанный программный код см. в приложении Б.

Заключение

Была написана программу на языке С, которая считывает изображение и обрабатывает его требуемым пользователем образом. Для этого были реализованы:

* Загрузка, хранение изображений из файла и запись изображения в файл;
* Ввод аргументов из коммандной строки;
* Функции для рисования на загруженном изображении.

Ввод аргументов из командной строки был осуществлен с помощью функции getopt\_long из стандартной библиотеки Си. Программа при любом желании пользователя не упадёт с ошибкой, а корректно завершит работу.

Список использованных источников

1. Язык программирования C / Керниган Брайан, Ритчи Деннис. СПб.: "Финансы и статистика", 2003.
2. Мануал по использованию libpng // libpng manual. URL: http://www.libpng.org/pub/png/libpng-1.2.5-manual.html (дата обращения: 25.04.2024)

Приложение А

**ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ**

|  |  |
| --- | --- |
| Изображение input.xyz | Изображение input.png |
|  |  |

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| ./cw --input input.xyz --pattern circle --color 255.0.0 --ornament |  |
| ./cw --output test.png --color 145.155.157 --border\_color 255.0.0 --filled\_rects --thickness 4 input.png |  |
| . /cw --output input.png --color\_replace --old\_color 39.32.24 --new\_color 1.1.1 input.png | Error: Input and output files cannot have the same name |
| ./cw --copy --left\_up -100.100 --right\_down 200.300 --dest\_left\_up 35.48 input.xyz |  |

Приложение Б

**Исходный код программы**

Файл main.c:

#include "structures.h"

#include "task\_handler.h"

#include "file\_handler.h"

#include "preparation\_handler.h"

/\*\*

\* @brief Main function to handle command-line arguments and process image tasks.

\*

\* @param argc: The number of command-line arguments.

\* @param argv: An array of strings containing the command-line arguments.

\*

\* @return int An integer representing the exit status of the program.

\*/

int main(int argc, char \*argv[]) {

/\* Initialize options struct with default values. \*/

Options options = {NULL};

/\* Set default output file name. \*/

options.output\_file = "out.png";

/\* Parse command-line arguments. \*/

handle\_arguments(argc, argv, &options);

/\* Initialize Png structure to hold information about the input PNG file. \*/

Png image;

/\* Read the input PNG file. \*/

read\_png\_file(options.input\_file, &image);

/\* Process tasks based on the provided options. \*/

task\_switcher(options, &image);

/\* Write the modified image to the output PNG file. \*/

write\_png\_file(options.output\_file, &image);

return 0;

}

Файл preparation\_handler.c:

#include "errors.h"

#include "structures.h"

#include "task\_handler.h"

/\*\*

\* @brief Handles command-line arguments passed to the program and populates the Options structure accordingly.

\*

\* @param argc An integer representing the number of command-line arguments.

\* @param argv An array of strings containing the command-line arguments.

\* @param options A pointer to the Options structure where the parsed arguments will be stored.

\*/

void handle\_arguments(int argc, char \*argv[], Options \*options) {

opterr = 0;

const char\* short\_options = "hi:o:";

const struct option long\_options[] = {

{"help", no\_argument, NULL, 'h'},

{"input", required\_argument, NULL, 'i'},

{"output", required\_argument, NULL, 'o'},

{"copy", no\_argument, NULL, 256},

{"color\_replace", no\_argument, NULL, 257},

{"ornament", no\_argument, NULL, 258},

{"filled\_rects", no\_argument, NULL, 259},

{"left\_up", required\_argument, NULL, 260},

{"right\_down", required\_argument, NULL, 261},

{"dest\_left\_up", required\_argument, NULL, 262},

{"old\_color", required\_argument, NULL, 263},

{"new\_color", required\_argument, NULL, 264},

{"pattern", required\_argument, NULL, 265},

{"color", required\_argument, NULL, 266},

{"thickness", required\_argument, NULL, 267},

{"count", required\_argument, NULL, 268},

{"border\_color", required\_argument, NULL, 269},

{"info", no\_argument, NULL, 270},

{NULL, 0, NULL, 0}

};

int res;

if (argc == 1) {

options->flag\_help = 1;

print\_help();

exit(EXIT\_SUCCESS);

return;

}

while ((res = getopt\_long(argc, argv, short\_options, long\_options, NULL)) != -1) {

switch (res) {

case 'h': /\* -h ot --help \*/

if (argc != 2) {

printf("Too many arguments for --help (-h)\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

options->flag\_help = 1;

break;

case 'i': /\* -i or --input \*/

options->input\_file = optarg;

options->flag\_input = 1;

break;

case 'o': /\* -o or --output \*/

options->output\_file = optarg;

options->flag\_output = 1;

break;

case 256: /\* --copy \*/

if (options->flag\_info || options->flag\_color\_replace || options->flag\_ornament || options->flag\_filled\_rects) {

printf("Error: Cannot use more than one function simultaneously\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

options->flag\_copy = 1;

break;

case 257: /\* --color\_replace \*/

if (options->flag\_info || options->flag\_copy || options->flag\_ornament || options->flag\_filled\_rects) {

printf("Error: Cannot use more than one function simultaneously\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

options->flag\_color\_replace = 1;

break;

case 258: /\* --ornament \*/

if (options->flag\_info || options->flag\_copy || options->flag\_color\_replace || options->flag\_filled\_rects) {

printf("Error: Cannot use more than one function simultaneously\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

options->flag\_ornament = 1;

break;

case 259: /\* --filled\_rects \*/

if (options->flag\_info || options->flag\_copy || options->flag\_ornament || options->flag\_color\_replace) {

printf("Error: Cannot use more than one function simultaneously\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

options->flag\_filled\_rects = 1;

break;

case 260: /\* --left\_up \*/

options->left\_up\_value = optarg;

options->flag\_left\_up = 1;

break;

case 261: /\* --right\_down \*/

options->right\_down\_value = optarg;

options->flag\_right\_down = 1;

break;

case 262: /\* --dest\_left\_up \*/

options->dest\_left\_up\_value = optarg;

options->flag\_dest\_left\_up = 1;

break;

case 263: /\* --old\_color \*/

options->flag\_old\_color = 1;

options->old\_color\_value = optarg;

break;

case 264: /\* --new\_color \*/

options->flag\_new\_color = 1;

options->new\_color\_value = optarg;

break;

case 265: /\* --pattern \*/

options->flag\_pattern = 1;

options->pattern\_value = optarg;

break;

case 266: /\* --color \*/

options->flag\_color = 1;

options->color\_value = optarg;

break;

case 267: /\* --thickness \*/

options->flag\_thickness = 1;

options->thickness\_value = optarg;

break;

case 268: /\* --count \*/

options->flag\_count = 1;

options->count\_value = optarg;

break;

case 269: /\* --border\_color \*/

options->flag\_border\_color = 1;

options->border\_color\_value = optarg;

break;

case 270: /\* --info \*/

options->flag\_info = 1;

break;

case '?':

default:

printf("Error: Unknown option or missing argument\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

break;

}

}

/\* No function provided \*/

if (!options->flag\_info && !options->flag\_help && !options->flag\_copy && !options->flag\_color\_replace && !options->flag\_ornament && !options->flag\_filled\_rects) {

printf("Error: No function provided\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

/\* -h or --help \*/

if (options->flag\_help) {

print\_help();

exit(EXIT\_SUCCESS);

}

/\* Not enough arguments for --copy \*/

if (options->flag\_copy && (!options->flag\_left\_up || !options->flag\_right\_down || !options->flag\_dest\_left\_up)) {

printf("Error: Insufficient arguments for --copy\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

/\* Not enough arguments for --color\_replace \*/

if (options->flag\_color\_replace && (!options->flag\_old\_color || !options->flag\_new\_color)) {

printf("Error: Insufficient arguments for --color\_replace\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

/\* Not enough arguments for --ornament \*/

if (options->flag\_ornament) {

if (!options->flag\_pattern || !options->flag\_color) {

printf("Error: Insufficient arguments for --ornament\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

if (strcmp("rectangle", options->pattern\_value) == 0 || strcmp("semicircles", options->pattern\_value) == 0) {

if (!options->flag\_thickness || !options->flag\_count) {

printf("Error: Insufficient arguments for --ornament\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

}

else if (strcmp("circle", options->pattern\_value) == 0) {

options->count\_value = "1";

options->thickness\_value = "1";

}

}

/\* Not enough arguments for --filled\_rects \*/

if (options->flag\_filled\_rects && (!options->flag\_border\_color || !options->flag\_color || !options->flag\_thickness)) {

printf("Error: Insufficient arguments for --filled\_rects\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

/\* Getting last argument (input file) or checking too many arguments \*/

if (!options->flag\_input) {

if (optind == argc - 1) {

options->input\_file = argv[argc - 1];

} else if (optind < argc - 1) {

printf("Error: Too many arguments\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

} else {

printf("Error: No input file provided\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

} else {

if (optind <= argc - 1) {

printf("Error: Too many arguments\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

}

/\* Input file has the same name as output file \*/

if (strcmp(options->input\_file, options->output\_file) == 0) {

printf("Error: Input and output files cannot have the same name\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

}

/\*\*

\* @brief Processes color provided as a string and returns it as an integer array.

\*

\* @param string\_color A string representing color in the format "R.G.B".

\* @return int\* An integer array containing the red, green, and blue components of the color.

\* NULL if the input string is invalid or if memory allocation fails.

\*/

int\* process\_color(char\* string\_color) {

/\* Takes color as "255.0.0" and returns as {255, 0, 0} \*/

int index = 0;

/\* If color starts or ends with '.' \*/

if (string\_color[strlen(string\_color)-1] == '.' || string\_color[0] == '.'){

return NULL;

}

char \*token = strtok(string\_color, ".");

int \*arr = malloc(sizeof(int)\*3);

if (arr == NULL) {

printf("Error: Can not allocate memory for array of colors\n");

exit(ERR\_MEMORY\_ALLOCATION\_FAILURE);

}

while (token != NULL && index < 3) {

arr[index++] = atoi(token);

token = strtok(NULL, ".");

}

/\* If there are less than 3 numbers or one of them are invalid \*/

if (token != NULL || index != 3 || arr[0] > 255 || arr[0] < 0 || arr[1] > 255 || arr[1] < 0 || arr[2] > 255 || arr[2] < 0){

return NULL;

}

return arr;

}

/\*\*

\* @brief Processes coordinates provided as a string and returns them as an integer array.

\*

\* @param string\_coordinates A string representing coordinates in the format "X.Y".

\* @return int\* An integer array containing the X and Y coordinates.

\* NULL if the input string is invalid or if memory allocation fails.

\*/

int\* process\_coordinates(char\* string\_coordinates){

/\* Takes coordinates as "100.200" and returns as {100, 200} \*/

int index = 0;

/\* If coordinates starts or ends with '.' \*/

if (string\_coordinates[strlen(string\_coordinates)-1] == '.' || string\_coordinates[0] == '.'){

return NULL;

}

char \*token = strtok(string\_coordinates, ".");

int \*arr = malloc(sizeof(int)\*2);

if (arr == NULL) {

printf("Error: Can not allocate memory for array of coordinates\n");

exit(ERR\_MEMORY\_ALLOCATION\_FAILURE);

}

while (token != NULL && index < 2) {

arr[index++] = atoi(token);

token = strtok(NULL, ".");

}

/\* If there are less than 2 numbers \*/

if (token != NULL || index != 2){

return NULL;

}

return arr;

}

Файл file\_handler.c:

#include "errors.h"

#include "structures.h"

/\*\*

\* @brief Reads a PNG file and stores its information and pixel data in a Png structure.

\*

\* @param file\_name A string representing the file name/path of the PNG image to be read.

\* @param image A pointer to the Png structure where the image data and information will be stored.

\*/

void read\_png\_file(char \*file\_name, Png \*image) {

int y;

char header[8];

/\* Open file \*/

FILE \*fp = fopen(file\_name, "rb");

if (!fp) {

printf("Error: Can not read file %s\n", file\_name);

exit(ERR\_FILE\_NOT\_FOUND);

}

/\* Read first 8 bytes to verify PNG file \*/

fread(header, 1, 8, fp);

if (png\_sig\_cmp((const unsigned char \*)header, 0, 8)) {

printf("Error: %s probably is not a PNG file\n", file\_name);

fclose(fp);

exit(ERR\_FILE\_READ\_ERROR);

}

/\* Create PNG read structure \*/

image->png\_ptr = png\_create\_read\_struct(PNG\_LIBPNG\_VER\_STRING, NULL, NULL, NULL);

if (!image->png\_ptr) {

printf("Error: Can not create PNG struct\n");

fclose(fp);

exit(ERR\_FILE\_READ\_ERROR);

}

/\* Create PNG info structure \*/

image->info\_ptr = png\_create\_info\_struct(image->png\_ptr);

if (!image->info\_ptr) {

printf("Error: Can not create PNG info struct\n");

fclose(fp);

png\_destroy\_read\_struct(&image->png\_ptr, NULL, NULL);

exit(ERR\_FILE\_READ\_ERROR);

}

/\* Set up error handling \*/

if (setjmp(png\_jmpbuf(image->png\_ptr))) {

printf("Error: Unknown\n");

fclose(fp);

png\_destroy\_read\_struct(&image->png\_ptr, &image->info\_ptr, NULL);

exit(ERR\_FILE\_READ\_ERROR);

}

/\* Initialize IO \*/

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);

/\* Check if color type is RGB \*/

if (png\_get\_color\_type(image->png\_ptr, image->info\_ptr) != PNG\_COLOR\_TYPE\_RGB) {

printf("Error: Not RGB color type in the file\n");

exit(ERR\_FILE\_READ\_ERROR);

}

/\* Allocate memory for image rows \*/

image->row\_pointers = malloc(sizeof(png\_bytep) \* image->height);

if (image->row\_pointers == NULL) {

printf("Error: Can not allocate memory for image->row\_pointers while reading\n");

exit(ERR\_MEMORY\_ALLOCATION\_FAILURE);

}

for (y = 0; y < image->height; y++) {

image->row\_pointers[y] = malloc(png\_get\_rowbytes(image->png\_ptr, image->info\_ptr));

if (image->row\_pointers[y] == NULL){

printf("Error: Can not allocate memory for pixel while reading\n");

exit(ERR\_MEMORY\_ALLOCATION\_FAILURE);

}

}

/\* Read image rows \*/

png\_read\_image(image->png\_ptr, image->row\_pointers);

/\* Close file \*/

fclose(fp);

}

/\*\*

\* @brief Writes a PNG image to a file.

\*

\* @param file\_name A string representing the file name/path where the PNG image will be saved.

\* @param image A pointer to the Png structure containing information about the PNG image.

\*/

void write\_png\_file(char \*file\_name, Png \*image) {

/\* Open file \*/

FILE \*fp = fopen(file\_name, "wb");

if (!fp) {

printf("Error: Can not create file: %s\n", file\_name);

exit(ERR\_FILE\_WRITE\_ERROR);

}

/\* Create PNG write structure \*/

png\_structp png\_ptr = png\_create\_write\_struct(PNG\_LIBPNG\_VER\_STRING, NULL, NULL, NULL);

if (!png\_ptr) {

printf("Error: Can not create PNG write struct\n");

fclose(fp);

exit(ERR\_FILE\_WRITE\_ERROR);

}

/\* Create PNG info structure \*/

png\_infop info\_ptr = png\_create\_info\_struct(png\_ptr);

if (!info\_ptr) {

printf("Error: Can not create PNG info struct while writing\n");

fclose(fp);

png\_destroy\_write\_struct(&png\_ptr, NULL);

exit(ERR\_FILE\_WRITE\_ERROR);

}

/\* Set up error handling \*/

if (setjmp(png\_jmpbuf(png\_ptr))) {

printf("Error: Unknown\n");

fclose(fp);

png\_destroy\_write\_struct(&png\_ptr, &info\_ptr);

exit(ERR\_FILE\_WRITE\_ERROR);

}

/\* Initialize IO \*/

png\_init\_io(png\_ptr, fp);

png\_set\_IHDR(png\_ptr, 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(png\_ptr, info\_ptr);

/\* Write image data \*/

png\_write\_image(png\_ptr, image->row\_pointers);

/\* Handle errors \*/

if (setjmp(png\_jmpbuf(png\_ptr))) {

printf("Error: Unknown\n");

fclose(fp);

png\_destroy\_write\_struct(&png\_ptr, &info\_ptr);

exit(ERR\_FILE\_WRITE\_ERROR);

}

/\* Finalize writing \*/

png\_write\_end(png\_ptr, NULL);

/\* Clean up \*/

for (int y = 0; y < image->height; y++)

free(image->row\_pointers[y]);

free(image->row\_pointers);

fclose(fp);

png\_destroy\_write\_struct(&png\_ptr, &info\_ptr);

}

Файл task\_handler.c:

#include "errors.h"

#include "structures.h"

#include "drawing\_handler.h"

#include "preparation\_handler.h"

/\*\*

\* @brief Prints the help message explaining the usage of the program and its options.

\*

\* This function does not return a value.

\*

\* The help message includes:

\* - Information about the course work and its creator.

\* - Usage syntax.

\* - Description of available options, including short and long forms, along with their corresponding explanations.

\*/

void print\_help() {

printf("Course work for option 5.16, created by Matvei Kolesnichenko.\n");

printf("Usage: ./program [OPTIONS] [input\_file]\n\n");

printf("Options:\n");

printf(" -h, --help Display this help message\n");

printf(" --info Print detailed information about the input PNG file\n");

printf(" -i, --input <filename> Specify the input PNG file\n");

printf(" -o, --output <filename> Specify the output PNG file (default: out.png)\n\n");

printf(" --copy Copy a specified region of the image\n");

printf(" --left\_up <x.y> Specify the coordinates of the top left corner of the source area\n");

printf(" --right\_down <x.y> Specify the coordinates of the bottom right corner of the source area\n");

printf(" --dest\_left\_up <x.y> Specify the coordinates of the top left corner of the destination area\n\n");

printf(" --color\_replace Replace all pixels of a specified color with another color\n");

printf(" --old\_color <r.g.b> Specify the color to be replaced\n");

printf(" --new\_color <r.g.b> Specify the color to replace with\n\n");

printf(" --ornament Create a patterned frame\n");

printf(" --pattern <rectangle|circle|semicircles>\n");

printf(" Specify the pattern of the frame\n");

printf(" --color <r.g.b> Specify the color of the frame\n");

printf(" --thickness <value> Specify the thickness of the frame\n");

printf(" --count <value> Specify the number of repetitions of the pattern\n\n");

printf(" --filled\_rects Find all filled rectangles of a specified color and draw an outline\n");

printf(" --color <r.g.b> Specify the color of the frame\n");

printf(" --border\_color <r.g.b> Specify the color of the outline\n");

printf(" --thickness <value> Specify the thickness of the outline\n");

}

/\*\*

\* @brief Prints information about a PNG image.

\*

\* @param image A pointer to the Png structure containing information about the PNG image.

\*

\* This function does not return a value.

\*

\* @note This function prints various details about the PNG image, including its width, height, color type, bit depth, and number of passes.

\* - The color type is printed as a string representation.

\* - Bit depth indicates the number of bits per sample or per channel in the image.

\* - Number of passes refers to the number of passes required for interlaced PNG images.

\*/

void print\_png\_info(Png \*image) {

printf("Image Width: %d\n", image->width);

printf("Image Height: %d\n", image->height);

printf("Color Type: ");

switch (image->color\_type) {

case PNG\_COLOR\_TYPE\_GRAY:

printf("Grayscale\n");

break;

case PNG\_COLOR\_TYPE\_RGB:

printf("RGB\n");

break;

case PNG\_COLOR\_TYPE\_PALETTE:

printf("Palette\n");

break;

case PNG\_COLOR\_TYPE\_GRAY\_ALPHA:

printf("Grayscale with Alpha\n");

break;

case PNG\_COLOR\_TYPE\_RGBA:

printf("RGB with Alpha\n");

break;

default:

printf("Unknown\n");

break;

}

printf("Bit Depth: %d\n", image->bit\_depth);

printf("Number of passes: %d\n", image->number\_of\_passes);

}

/\*\*

\* @brief Replaces all pixels of the specified old color with the new color.

\*

\* @param image A pointer to the Png structure representing the image.

\* @param old\_color A string representing the old color in the format "R,G,B".

\* @param new\_color A string representing the new color in the format "R,G,B".

\*

\* This function does not return a value.

\*/

void color\_replace(Png \*image, char\* old\_color, char\* new\_color) {

int x, y;

/\* Getting colors as arrays \*/

int\* old\_color\_values = process\_color(old\_color);

int\* new\_color\_values = process\_color(new\_color);

/\* Error handling \*/

if (!old\_color\_values || !new\_color\_values) {

printf("Error: Can not process color\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

/\* Nested loop to iterate through each pixel of the image \*/

for (y = 0; y < image->height; y++) {

png\_bytep row = image->row\_pointers[y];

for (x = 0; x < image->width; x++) {

png\_bytep ptr = &(row[x \* 3]); // Getting pixel

/\* Changing by colors \*/

if ((ptr[0] == old\_color\_values[0]) && (ptr[1] == old\_color\_values[1]) && (ptr[2] == old\_color\_values[2])) {

draw\_pixel(ptr, new\_color\_values);

}

}

}

/\* Clean up \*/

free(old\_color\_values);

free(new\_color\_values);

}

/\*\*

\* @brief Copies the specified area from the original image to a new structure, then copies it back to the original image at a different location.

\*

\* @param image A pointer to the Png structure representing the original image.

\* @param left\_up A string containing the coordinates of the top-left corner of the area to be copied in the format "x,y".

\* @param right\_down A string containing the coordinates of the bottom-right corner of the area to be copied in the format "x,y".

\* @param dest\_left\_up A string containing the coordinates of the top-left corner of the destination location in the original image for the copied area.

\*

\* This function does not return a value.

\*/

void copy\_area(Png \*image, char\* left\_up, char\* right\_down, char\* dest\_left\_up) {

int x, y;

int area\_x = 0, area\_y = 0;

/\* Getting coordinates as arrays \*/

int\* left\_up\_coordinates = process\_coordinates(left\_up);

int\* right\_down\_coordinates = process\_coordinates(right\_down);

int\* dest\_left\_up\_coordinates = process\_coordinates(dest\_left\_up);

/\* Error handling \*/

if (!left\_up\_coordinates || !right\_down\_coordinates || !dest\_left\_up\_coordinates){

printf("Error: Can not process coordinates\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

/\* Making structure and its information \*/

Area\* copied\_area = malloc(sizeof(Area));

if (copied\_area == NULL) {

printf("Error: Can not allocate memory for copied area\n");

exit(ERR\_MEMORY\_ALLOCATION\_FAILURE);

}

copied\_area->height = abs(right\_down\_coordinates[1] - left\_up\_coordinates[1]) + 1;

copied\_area->width = abs(right\_down\_coordinates[0] - left\_up\_coordinates[0]) + 1;

copied\_area->row\_pointers = malloc(sizeof(png\_bytep) \* copied\_area->height);

if (copied\_area->row\_pointers == NULL) {

printf("Error: Can not allocate memory for copied area row\_pointers\n");

exit(ERR\_MEMORY\_ALLOCATION\_FAILURE);

}

/\* Allocating memory for each pixel \*/

for (y = 0; y < copied\_area->height; y++) {

copied\_area->row\_pointers[y] = malloc(sizeof(png\_byte) \* copied\_area->width \* 3);

if (copied\_area->row\_pointers[y] == NULL) {

printf("Error: Can not allocate memory for copied area pixel\n");

exit(ERR\_MEMORY\_ALLOCATION\_FAILURE);

}

}

/\* Switching left and right if needed \*/

if (left\_up\_coordinates[0] > right\_down\_coordinates[0]){

int tmp[2];

tmp[0] = right\_down\_coordinates[0];

tmp[1] = right\_down\_coordinates[1];

right\_down\_coordinates[0] = left\_up\_coordinates[0];

right\_down\_coordinates[1] = left\_up\_coordinates[1];

left\_up\_coordinates[0] = tmp[0];

left\_up\_coordinates[1] = tmp[1];

}

/\* Copying area from original image to structure \*/

for (y = left\_up\_coordinates[1]; y <= right\_down\_coordinates[1]; y++) {

for (x = left\_up\_coordinates[0]; x <= right\_down\_coordinates[0]; x++) {

/\* Handling inappropriate coordinate \*/

if (x < 0 || x >= image->width || y < 0 || y >= image->height) {

copied\_area->row\_pointers[area\_y][area\_x\*3] = 0;

copied\_area->row\_pointers[area\_y][area\_x\*3 + 1] = 0;

copied\_area->row\_pointers[area\_y][area\_x\*3 + 2] = 0;

} else {

/\* Copying every pixel by R, G, B \*/

copied\_area->row\_pointers[area\_y][area\_x \* 3] = image->row\_pointers[y][x \* 3];

copied\_area->row\_pointers[area\_y][area\_x \* 3 + 1] = image->row\_pointers[y][x \* 3 + 1];

copied\_area->row\_pointers[area\_y][area\_x \* 3 + 2] = image->row\_pointers[y][x \* 3 + 2];

}

area\_x++;

}

area\_x = 0;

area\_y++;

}

area\_x = 0;

area\_y = 0;

/\* Copying area from structure back to original image \*/

for (y = dest\_left\_up\_coordinates[1]; y < dest\_left\_up\_coordinates[1] + copied\_area->height; y++) {

for (x = dest\_left\_up\_coordinates[0]; x < dest\_left\_up\_coordinates[0] + copied\_area->width; x++) {

/\* Handling inappropriate coordinate \*/

if (x < 0 || x >= image->width || y < 0 || y >= image->height) {

continue;

}

if (copied\_area->row\_pointers[area\_y][area\_x \* 3] != 0 && copied\_area->row\_pointers[area\_y][area\_x \* 3 + 1] != 0 && copied\_area->row\_pointers[area\_y][area\_x \* 3 + 2] != 0) {

/\* Copying every pixel by R, G, B \*/

image->row\_pointers[y][x \* 3] = copied\_area->row\_pointers[area\_y][area\_x \* 3];

image->row\_pointers[y][x \* 3 + 1] = copied\_area->row\_pointers[area\_y][area\_x \* 3 + 1];

image->row\_pointers[y][x \* 3 + 2] = copied\_area->row\_pointers[area\_y][area\_x \* 3 + 2];

}

area\_x++;

}

area\_x = 0;

area\_y++;

}

/\* Clean up \*/

free(left\_up\_coordinates);

free(right\_down\_coordinates);

free(dest\_left\_up\_coordinates);

for (y = 0; y < copied\_area->height; y++)

free(copied\_area->row\_pointers[y]);

free(copied\_area->row\_pointers);

}

/\*\*

\* @brief Finds all filled rectangles in the image and draws borders around them.

\*

\* @param image A pointer to the Png structure representing the image.

\* @param string\_color A string representing the color of the filled rectangles in the format "rrr.ggg.bbb".

\* @param string\_border\_color A string representing the color of the border in the format "rrr.ggg.bbb".

\* @param thickness A string representing the thickness of the border.

\*

\* This function does not return a value.

\*/

void filled\_rects(Png \*image, char\* string\_color, char\* string\_border\_color, char\* thickness) {

/\* Allocate memory for tracking visited pixels \*/

int\*\* visited = malloc(sizeof(int\*) \* image->height);

if (visited == NULL) {

printf("Error: Can not allocate memory for rectangles visited pixels\n");

exit(ERR\_MEMORY\_ALLOCATION\_FAILURE);

}

for (int i = 0; i < image->height; i++) {

visited[i] = malloc(sizeof(int) \* image->width);

if (visited[i] == NULL) {

printf("Error: Can not allocate memory for rectangles visited pixels\n");

exit(ERR\_MEMORY\_ALLOCATION\_FAILURE);

}

for (int j = 0; j < image->width; j++) {

visited[i][j] = 0;

}

}

/\* Getting color as array \*/

int\* color\_values = process\_color(string\_color);

/\* Error handling: Cannot process rectangle color \*/

if (!color\_values) {

printf("Error: Can not process rectangle color\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

/\* Processing border color \*/

int\* border\_color = process\_color(string\_border\_color);

/\* Error handling: Cannot process border color \*/

if (!border\_color) {

printf("Error: Can not process border color\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

/\* Nested loop to iterate through all pixels \*/

for (int y = 0; y < image->height; y++) {

png\_bytep row = image->row\_pointers[y];

for (int x = 0; x < image->width; x++) {

/\* Skip visited or other color pixels \*/

if (visited[y][x] || !(row[x \* 3] == color\_values[0] && row[x \* 3 + 1] == color\_values[1] && row[x \* 3 + 2] == color\_values[2])) {

continue;

}

int start\_x = x;

int start\_y = y;

int end\_x = start\_x;

int end\_y = start\_y;

/\* Finding end horizontally \*/

while (end\_x < image->width && visited[end\_y][end\_x] == 0 && row[end\_x \* 3] == color\_values[0] && row[end\_x \* 3 + 1] == color\_values[1] && row[end\_x \* 3 + 2] == color\_values[2]) {

end\_x++;

}

/\* Finding end vertically \*/

while (end\_y < image->height) {

int found\_different\_color = 0;

for (int i = start\_x; i < end\_x; i++) {

if (visited[end\_y][i] || !(image->row\_pointers[end\_y][i \* 3] == color\_values[0] && image->row\_pointers[end\_y][i \* 3 + 1] == color\_values[1] && image->row\_pointers[end\_y][i \* 3 + 2] == color\_values[2])) {

found\_different\_color = 1;

break;

}

}

if (found\_different\_color) {

break;

}

end\_y++;

}

draw\_border(image, start\_x, start\_y, end\_x - 1, end\_y - 1, border\_color, thickness);

/\* Marking visited pixels \*/

for (int i = start\_y; i < end\_y; i++) {

for (int j = start\_x; j < end\_x; j++) {

visited[i][j] = 1;

}

}

}

}

/\* Clean up \*/

free(color\_values);

free(border\_color);

for (int y = 0; y < image->height; y++)

free(visited[y]);

free(visited);

}

/\*\*

\* @brief Draws an ornament pattern on the given image.

\*

\* @param image A pointer to the Png structure representing the image.

\* @param pattern A string specifying the type of ornament pattern ("rectangle", "circle", "semicircles").

\* @param string\_color A string representing the color of the ornament in the format "rrr.ggg.bbb".

\* @param thickness A string representing the thickness of the ornament.

\* @param count A string representing the number of ornaments to be drawn.

\*

\* This function does not return a value.

\*/

void ornament(Png \*image, char\* pattern, char\* string\_color, char\* thickness, char\* count) {

/\* Getting color as array \*/

int\* color\_values = process\_color(string\_color);

/\* Error handling: Cannot process ornament color \*/

if (!color\_values) {

printf("Error: Can not process ornament color\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

/\* Getting thickness as integer \*/

int ornament\_thickness = atoi(thickness);

/\* Error handling: Ornament thickness is not a positive integer \*/

if (ornament\_thickness <= 0) {

printf("Error: Ornament thickness is not a positive integer\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

/\* Getting count as integer \*/

int ornament\_count = atoi(count);

/\* Error handling: Ornament count is not a positive integer \*/

if (ornament\_count <= 0) {

printf("Error: Ornament count is not a positive integer\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

/\* Rectangle pattern \*/

if (strcmp(pattern, "rectangle") == 0){

rectangle\_ornament(image, ornament\_thickness, ornament\_count, color\_values, thickness);

/\* Circle pattern \*/

} else if (strcmp(pattern, "circle") == 0) {

circle\_ornament(image, color\_values);

/\* Semicircles pattern \*/

} else if (strcmp(pattern, "semicircles") == 0){

semicircles\_ornament(image, ornament\_thickness, ornament\_count, color\_values);

}

/\* Unknown pattern \*/

else {

printf("Error: Unknown pattern\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

/\* Clean up \*/

free(color\_values);

}

void outside\_ornament(Png \*image, char\* thickness, char\* string\_color){

int x,y;

/\* Getting color as array \*/

int\* color\_values = process\_color(string\_color);

/\* Error handling: Cannot process border color \*/

if (!color\_values) {

printf("Error: Can not process border color\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

/\* Getting thickness as integer \*/

int border\_thickness = atoi(thickness);

/\* Error handling: border thickness is not a positive integer \*/

if (border\_thickness <= 0) {

printf("Error: border thickness is not a positive integer\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

int area\_x = 0;

int area\_y = 0;

/\* Making structure and its information \*/

Area\* copied\_area = malloc(sizeof(Area));

if (copied\_area == NULL) {

printf("Error: Can not allocate memory for copied area\n");

exit(ERR\_MEMORY\_ALLOCATION\_FAILURE);

}

copied\_area->height = image->height;

copied\_area->width = image->width;

copied\_area->row\_pointers = malloc(sizeof(png\_bytep) \* copied\_area->height);

if (copied\_area->row\_pointers == NULL) {

printf("Error: Can not allocate memory for copied area row\_pointers\n");

exit(ERR\_MEMORY\_ALLOCATION\_FAILURE);

}

/\* Allocating memory for each pixel \*/

for (y = 0; y < copied\_area->height; y++) {

copied\_area->row\_pointers[y] = malloc(sizeof(png\_byte) \* copied\_area->width \* 3);

if (copied\_area->row\_pointers[y] == NULL) {

printf("Error: Can not allocate memory for copied area pixel\n");

exit(ERR\_MEMORY\_ALLOCATION\_FAILURE);

}

}

/\* Copying area from original image to structure \*/

for (y = 0; y < image->height; y++) {

for (x = 0; x < image->width; x++) {

/\* Handling inappropriate coordinate \*/

if (x < 0 || x >= image->width || y < 0 || y >= image->height) {

copied\_area->row\_pointers[area\_y][area\_x\*3] = 0;

copied\_area->row\_pointers[area\_y][area\_x\*3 + 1] = 0;

copied\_area->row\_pointers[area\_y][area\_x\*3 + 2] = 0;

} else {

/\* Copying every pixel by R, G, B \*/

copied\_area->row\_pointers[area\_y][area\_x \* 3] = image->row\_pointers[y][x \* 3];

copied\_area->row\_pointers[area\_y][area\_x \* 3 + 1] = image->row\_pointers[y][x \* 3 + 1];

copied\_area->row\_pointers[area\_y][area\_x \* 3 + 2] = image->row\_pointers[y][x \* 3 + 2];

}

area\_x++;

}

area\_x = 0;

area\_y++;

}

image->height = image->height + 2\*border\_thickness;

image->width = image->width + 2\*border\_thickness;

image->row\_pointers = malloc(sizeof(png\_bytep) \* image->height);

if (image->row\_pointers == NULL) {

printf("Error: Can not allocate memory for image->row\_pointers while reading\n");

exit(ERR\_MEMORY\_ALLOCATION\_FAILURE);

}

for (y = 0; y < image->height; y++) {

image->row\_pointers[y] = malloc(sizeof(png\_byte)\*3\*image->width);

if (image->row\_pointers[y] == NULL){

printf("Error: Can not allocate memory for pixel while reading\n");

exit(ERR\_MEMORY\_ALLOCATION\_FAILURE);

}

}

for (y = 0; y < image->height; y++) {

for (x = 0; x < image->width; x++) {

image->row\_pointers[y][x \* 3] = color\_values[0];

image->row\_pointers[y][x \* 3 + 1] = color\_values[1];

image->row\_pointers[y][x \* 3 + 2] = color\_values[2];

}

}

area\_x = 0;

area\_y = 0;

/\* Copying area from structure back to original image \*/

for (y = border\_thickness; y < border\_thickness + copied\_area->height; y++) {

for (x = border\_thickness; x < border\_thickness + copied\_area->width; x++) {

/\* Handling inappropriate coordinate \*/

if (x < 0 || x >= image->width || y < 0 || y >= image->height) {

continue;

}

/\* Copying every pixel by R, G, B \*/

image->row\_pointers[y][x \* 3] = copied\_area->row\_pointers[area\_y][area\_x \* 3];

image->row\_pointers[y][x \* 3 + 1] = copied\_area->row\_pointers[area\_y][area\_x \* 3 + 1];

image->row\_pointers[y][x \* 3 + 2] = copied\_area->row\_pointers[area\_y][area\_x \* 3 + 2];

area\_x++;

}

area\_x = 0;

area\_y++;

}

/\* Clean up \*/

free(color\_values);

for (y = 0; y < copied\_area->height; y++)

free(copied\_area->row\_pointers[y]);

free(copied\_area->row\_pointers);

}

/\*\*

\* @brief Handles task switching based on provided options.

\*

\* @param options Options structure containing flags and values for various tasks.

\* @param image Pointer to the Image structure representing the image.

\*

\* This function does not return a value.

\*/

void task\_switcher(Options options, Png \*image) {

if (options.flag\_info) {

print\_png\_info(image);

exit(EXIT\_SUCCESS);

}

if (options.flag\_outside\_ornament) {

outside\_ornament(image, options.thickness\_value, options.color\_value);

}

if (options.flag\_copy) {

copy\_area(image, options.left\_up\_value, options.right\_down\_value, options.dest\_left\_up\_value);

}

if (options.flag\_color\_replace) {

color\_replace(image, options.old\_color\_value, options.new\_color\_value);

}

if (options.flag\_ornament) {

ornament(image, options.pattern\_value, options.color\_value, options.thickness\_value, options.count\_value);

}

if (options.flag\_filled\_rects) {

filled\_rects(image, options.color\_value, options.border\_color\_value, options.thickness\_value);

}

}

Файл drawing\_handler.c:

#include "errors.h"

#include "structures.h"

#include "preparation\_handler.h"

/\*\*

\* @brief Draws a single pixel with the specified color values.

\*

\* @param ptr Pointer to the pixel in the image.

\* @param color\_values Array containing the RGB values of the pixel color.

\*/

void draw\_pixel(png\_bytep ptr, int\* color\_values) {

ptr[0] = color\_values[0];

ptr[1] = color\_values[1];

ptr[2] = color\_values[2];

}

/\*\*

\* @brief Draws a border around the specified rectangle in the image.

\*

\* @param image Pointer to the Png structure representing the image.

\* @param x1 The x-coordinate of the top-left corner of the rectangle.

\* @param y1 The y-coordinate of the top-left corner of the rectangle.

\* @param x2 The x-coordinate of the bottom-right corner of the rectangle.

\* @param y2 The y-coordinate of the bottom-right corner of the rectangle.

\* @param border\_color Array containing the RGB values of the border color.

\* @param thickness String representing the thickness of the border.

\*/

void draw\_border(Png \*image, int x1, int y1, int x2, int y2, int\* border\_color, char\* thickness) {

/\* Convert thickness string to integer \*/

int border\_thickness = atoi(thickness);

if (border\_thickness <= 0) {

printf("Error: Border thickness is not a positive integer\n");

exit(ERR\_INSUFFICIENT\_ARGUMENTS);

}

/\* Draw horizontal lines \*/

for (int t = 1; t <= border\_thickness; t++) {

/\* Draw upper horizontal line \*/

int y = y1 - t;

if (y >= 0 && y < image->height) {

png\_bytep row = image->row\_pointers[y];

for (int x = x1 - t; x <= x2 + t; x++) {

if (x >= 0 && x < image->width) {

png\_bytep px = &(row[x \* 3]);

draw\_pixel(px, border\_color);

}

}

}

/\* Draw lower horizontal line \*/

y = y2 + t;

if (y >= 0 && y < image->height) {

png\_bytep row = image->row\_pointers[y];

for (int x = x1 - t; x <= x2 + t; x++) {

if (x >= 0 && x < image->width) {

png\_bytep px = &(row[x \* 3]);

draw\_pixel(px, border\_color);

}

}

}

}

/\* Draw vertical lines \*/

for (int t = 1; t <= border\_thickness; t++) {

/\* Draw left vertical line \*/

int x = x1 - t;

if (x >= 0 && x < image->width) {

for (int y = y1 - t; y <= y2 + t; y++) {

if (y >= 0 && y < image->height) {

png\_bytep px = &(image->row\_pointers[y][(x \* 3)]);

draw\_pixel(px, border\_color);

}

}

}

/\* Draw right vertical line \*/

x = x2 + t;

if (x >= 0 && x < image->width) {

for (int y = y1 - t; y <= y2 + t; y++) {

if (y >= 0 && y < image->height) {

png\_bytep px = &(image->row\_pointers[y][(x \* 3)]);

draw\_pixel(px, border\_color);

}

}

}

}

}

/\*\*

\* @brief Draws rectangle ornaments on the image.

\*

\* @param image Pointer to the Png structure representing the image.

\* @param ornament\_thickness Thickness of the ornament rectangles.

\* @param ornament\_count Number of ornament rectangles to draw.

\* @param color\_values Array containing the RGB values of the ornament color.

\* @param thickness String representing the thickness of the border.

\*/

void rectangle\_ornament(Png \*image, int ornament\_thickness, int ornament\_count, int\* color\_values, char\* thickness) {

int x1, y1, x2, y2;

x1 = ornament\_thickness;

y1 = ornament\_thickness;

x2 = image->width - ornament\_thickness - 1;

y2 = image->height - ornament\_thickness - 1;

for (int i = 0; i < ornament\_count; i++){

draw\_border(image, x1, y1, x2, y2, color\_values, thickness);

x1 += ornament\_thickness \* 2;

y1 += ornament\_thickness \* 2;

x2 -= ornament\_thickness \* 2;

y2 -= ornament\_thickness \* 2;

/\* Check if rectangles can fit \*/

if (x1 >= x2 || y1 >= y2){

printf("Warning: Rectangles that cannot fit will be skipped\n");

break;

}

}

}

/\*\*

\* @brief Draws a circle ornament on the image.

\*

\* @param image Pointer to the Png structure representing the image.

\* @param color\_values Array containing the RGB values of the ornament color.

\*/

void circle\_ornament(Png \*image, int\* color\_values) {

int centerX = image->width / 2;

int centerY = image->height / 2;

int radius = (centerX < centerY) ? centerX : centerY;

for (int y = 0; y < image->height; y++) {

for (int x = 0; x < image->width; x++) {

double distance = sqrt(pow(x - centerX, 2) + pow(y - centerY, 2));

if (distance <= radius) {

continue;

}

png\_bytep ptr = &(image->row\_pointers[y][x \* 3]);

draw\_pixel(ptr, color\_values);

}

}

}

/\*\*

\* @brief Draws semicircle ornaments on the image.

\*

\* @param image Pointer to the Png structure representing the image.

\* @param ornament\_thickness Thickness of the semicircle ornaments.

\* @param ornament\_count Number of semicircle ornaments to draw.

\* @param color\_values Array containing the RGB values of the ornament color.

\*/

void semicircles\_ornament(Png \*image, int ornament\_thickness, int ornament\_count, int\* color\_values) {

int radiusX = ceil((double)(image->width - ornament\_count \* ornament\_thickness) / (2 \* ornament\_count));

int radiusY = ceil((double)(image->height - ornament\_count \* ornament\_thickness) / (2 \* ornament\_count));

int centerX = radiusX + ceil(ornament\_thickness/2);

for (int i = 0; i < ornament\_count; i++){

for (int x = centerX - radiusX - ceil(ornament\_thickness/2); x < centerX + radiusX + ornament\_thickness && x < image->width && x >= 0; x++){

/\* Draw upper semicircles \*/

for (int y = 0; y < radiusX + ornament\_thickness && y < image->height && y >= 0; y++){

int centerY = 0;

double distance = sqrt(pow(x - centerX, 2) + pow(y - centerY, 2));

if (distance < radiusX || distance > radiusX + ornament\_thickness) {

continue;

}

png\_bytep ptr = &(image->row\_pointers[y][x \* 3]);

draw\_pixel(ptr, color\_values);

}

/\* Draw lower semicircles \*/

for (int y = image->height - 1; y > image->height - 1 - radiusX - ornament\_thickness && y >= 0 && y < image->height; y--){

int centerY = image->height - 1;

double distance = sqrt(pow(x - centerX, 2) + pow(y - centerY, 2));

if (distance < radiusX || distance > radiusX + ornament\_thickness) {

continue;

}

png\_bytep ptr = &(image->row\_pointers[y][x \* 3]);

draw\_pixel(ptr, color\_values);

}

}

centerX += 2 \* radiusX + ornament\_thickness;

}

int centerY = radiusY + ceil(ornament\_thickness/2);

for (int i = 0; i < ornament\_count; i++){

for (int y = centerY - radiusY - ceil(ornament\_thickness/2); y < centerY + radiusY + ornament\_thickness && y < image->height && y >= 0; y++){

/\* Draw left semicircles \*/

for (int x = 0; x < radiusY + ornament\_thickness && x < image->width && x >= 0; x++){

int centerX = 0;

double distance = sqrt(pow(x - centerX, 2) + pow(y - centerY, 2));

if (distance < radiusY || distance > radiusY + ornament\_thickness) {

continue;

}

png\_bytep ptr = &(image->row\_pointers[y][x \* 3]);

draw\_pixel(ptr, color\_values);

}

/\* Draw right semicircles \*/

for (int x = image->width - 1; x > image->width - 1 - radiusY - ornament\_thickness && x < image->width && x >= 0; x--){

int centerX = image->width - 1;

double distance = sqrt(pow(x - centerX, 2) + pow(y - centerY, 2));

if (distance < radiusY || distance > radiusY + ornament\_thickness) {

continue;

}

png\_bytep ptr = &(image->row\_pointers[y][x \* 3]);

draw\_pixel(ptr, color\_values);

}

}

centerY += 2 \* radiusY + ornament\_thickness;

}

}

Файл errors.h:

#ifndef ERRORS\_H

#define ERRORS\_H

/\* Error code indicating that the specified file could not be found. \*/

#define ERR\_FILE\_NOT\_FOUND 40

/\* Error code indicating an error occurred while reading the file. \*/

#define ERR\_FILE\_READ\_ERROR 41

/\* Error code indicating an error occurred while writing to the file. \*/

#define ERR\_FILE\_WRITE\_ERROR 42

/\* Error code indicating an error occurred while closing the file. \*/

#define ERR\_FILE\_CLOSE\_ERROR 43

/\* Error code indicating that insufficient command-line arguments were provided. \*/

#define ERR\_INSUFFICIENT\_ARGUMENTS 45

/\* Error code indicating a failure in memory allocation. \*/

#define ERR\_MEMORY\_ALLOCATION\_FAILURE 46

#endif

Файл structures.h:

#ifndef STRUCTURES\_H

#define STRUCTURES\_H

#include <png.h>

#include <stdio.h>

#include <stdlib.h>

#include <getopt.h>

#include <string.h>

#include <math.h>

/\*\*

\* @brief Structure representing a PNG image.

\*/

typedef struct Png {

int width; /\*\*< Width of the image in pixels \*/

int height; /\*\*< Height of the image in pixels \*/

png\_byte color\_type; /\*\*< Color type of the image (e.g., RGB, Grayscale) \*/

png\_byte bit\_depth; /\*\*< Bit depth of the image \*/

png\_structp png\_ptr; /\*\*< Pointer to the libpng structure for reading/writing PNG data \*/

png\_infop info\_ptr; /\*\*< Pointer to the libpng structure for storing PNG information \*/

int number\_of\_passes; /\*\*< Number of passes required for interlacing (typically used for progressive rendering) \*/

png\_bytep \*row\_pointers; /\*\*< Pointer to an array of pointers, each pointing to a row of image data \*/

} Png;

/\*\*

\* @brief Structure representing an area copied during the execution of the 'copy\_area' function.

\*/

typedef struct Area {

int width; /\*\*< Width of the area in pixels \*/

int height; /\*\*< Height of the area in pixels \*/

png\_bytep \*row\_pointers; /\*\*< Pointer to an array of pointers, each pointing to a row of area data \*/

} Area;

/\*\*

\* @brief Structure representing options provided to the program.

\*/

typedef struct {

char\* input\_file; /\*\*< Filename of the input PNG file \*/

char\* output\_file; /\*\*< Filename of the output PNG file \*/

int flag\_help; /\*\*< Flag indicating if the help message should be displayed \*/

int flag\_input; /\*\*< Flag indicating if the input file has been specified \*/

int flag\_output; /\*\*< Flag indicating if the output file has been specified \*/

int flag\_copy; /\*\*< Flag indicating if the 'copy' function should be executed \*/

int flag\_color\_replace; /\*\*< Flag indicating if the 'color\_replace' function should be executed \*/

int flag\_ornament; /\*\*< Flag indicating if the 'ornament' function should be executed \*/

int flag\_filled\_rects; /\*\*< Flag indicating if the 'filled\_rects' function should be executed \*/

int flag\_left\_up; /\*\*< Flag indicating if the top-left coordinate of the source area has been specified \*/

int flag\_right\_down; /\*\*< Flag indicating if the bottom-right coordinate of the source area has been specified \*/

int flag\_dest\_left\_up; /\*\*< Flag indicating if the top-left coordinate of the destination area has been specified \*/

int flag\_old\_color; /\*\*< Flag indicating if the old color for color replacement has been specified \*/

int flag\_new\_color; /\*\*< Flag indicating if the new color for color replacement has been specified \*/

int flag\_pattern; /\*\*< Flag indicating if the pattern for ornamentation has been specified \*/

int flag\_color; /\*\*< Flag indicating if the color for ornamentation or filled rectangles has been specified \*/

int flag\_thickness; /\*\*< Flag indicating if the thickness for ornamentation or filled rectangles has been specified \*/

int flag\_count; /\*\*< Flag indicating if the count for ornamentation has been specified \*/

int flag\_border\_color; /\*\*< Flag indicating if the border color for filled rectangles has been specified \*/

int flag\_info; /\*\*< Flag indicating if detailed information about the input PNG file should be printed \*/

char\* left\_up\_value; /\*\*< Value of the top-left coordinate of the source area \*/

char\* right\_down\_value; /\*\*< Value of the bottom-right coordinate of the source area \*/

char\* dest\_left\_up\_value; /\*\*< Value of the top-left coordinate of the destination area \*/

char\* old\_color\_value; /\*\*< Value of the old color for color replacement \*/

char\* new\_color\_value; /\*\*< Value of the new color for color replacement \*/

char\* pattern\_value; /\*\*< Value of the pattern for ornamentation \*/

char\* color\_value; /\*\*< Value of the color for ornamentation or filled rectangles \*/

char\* thickness\_value; /\*\*< Value of the thickness for ornamentation or filled rectangles \*/

char\* count\_value; /\*\*< Value of the count for ornamentation \*/

char\* border\_color\_value; /\*\*< Value of the border color for filled rectangles \*/

} Options;

#endif

Файл preparation\_handler.h:

#ifndef PREPARATION\_HANDLER\_H

#define PREPARATION\_HANDLER\_H

#include "structures.h"

void handle\_arguments(int argc, char \*argv[], Options \*options);

int\* process\_color(char\* string\_color);

int\* process\_coordinates(char\* string\_coordinates);

#endif

Файл file\_handler.h:

#ifndef FILE\_HANDLER\_H

#define FILE\_HANDLER\_H

#include "structures.h"

void read\_png\_file(char \*file\_name, Png \*image);

void write\_png\_file(char \*file\_name, Png \*image);

#endif

Файл task\_handler.h:

#ifndef TASK\_HANDLER\_H

#define TASK\_HANDLER\_H

#include "structures.h"

void print\_help();

void print\_png\_info(Png \*image);

void task\_switcher(Options options, Png \*image);

void color\_replace(Png \*image, char\* old\_color, char\* new\_color);

void copy\_area(Png \*image, char\* left\_up, char\* right\_down, char\* dest\_left\_up);

void filled\_rects(Png \*image, char\* string\_color, char\* string\_border\_color, char\* thickness);

void ornament(Png \*image, char\* pattern, char\* string\_color, char\* thickness, char\* count);

#endif

Файл drawing\_handler.h:

#ifndef DRAWING\_HANDLER\_H

#define DRAWING\_HANDLER\_H

#include "structures.h"

void draw\_pixel(png\_bytep ptr, int\* color\_values);

void draw\_border(Png \*image, int x1, int y1, int x2, int y2, int\* border\_color, char\* thickness);

void rectangle\_ornament(Png \*image, int ornament\_thickness, int ornament\_count, int\* color\_values, char\* thickness);

void circle\_ornament(Png \*image, int\* color\_values);

void semicircles\_ornament(Png \*image, int ornament\_thickness, int ornament\_count, int\* color\_values);

#endif

Файл Makefile:

CC = gcc

CFLAGS = -Wall -Wextra -std=c99

LDFLAGS = -lpng -lm

SRCDIR = src

INCDIR = include

BUILDDIR = build

DOCSDIR = docs

LATEXDIR = $(DOCSDIR)/latex

HTMLDIR = $(DOCSDIR)/html

SOURCES = $(wildcard $(SRCDIR)/\*.c)

OBJECTS = $(patsubst $(SRCDIR)/%.c,$(BUILDDIR)/%.o,$(SOURCES))

EXECUTABLE = cw

.PHONY: all clean docs

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)

$(CC) $^ -o $@ $(LDFLAGS)

$(BUILDDIR)/%.o: $(SRCDIR)/%.c

@mkdir -p $(BUILDDIR)

$(CC) $(CFLAGS) -I$(INCDIR) -c $< -o $@

addons:

mkdir -p $(DOCSDIR)/doxygen-awesome-css

git clone https://github.com/w3hhh-m/COURSEWORK-2SEM-ADDONS $(DOCSDIR)/doxygen-awesome-css

mv $(DOCSDIR)/doxygen-awesome-css/Doxyfile .

docs: addons

doxygen Doxyfile

cd $(LATEXDIR) && \

rm -f refman.tex && \

cp ../doxygen-awesome-css/refman.tex . && \

make && \

mv refman.pdf ../docs.pdf && \

cd .. && \

rm -rf latex

clean:

rm -f $(EXECUTABLE)

rm -rf $(BUILDDIR)

rm -rf $(DOCSDIR)

rm -f Doxyfile