

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_orname nt
 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 искомым 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 C. 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 `Doxugen`. 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
1. Структуры	9
1.1 Структура Png.....	9
1.2 Структура Area	9
1.3 Структура Options	10
2. Функции.....	11
2.1 Функция main	11
2.2 Подготовительные функции	11
2.3 Функции для работы с файлами	12
2.4 Функции выполнения заданий	12
2.5 Функции рисования.....	14
Заключение	15
Список использованных источников	16
Приложение А	17
Приложение Б.....	19

ВВЕДЕНИЕ

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

- Загрузку, хранение изображений из файла и запись изображения в файл;
- Ввод аргументов из командной 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 программа на языке C, 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 командной 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 С / 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
<pre>./cw --input input.xyz --pattern circle --color 255.0.0 --ornament</pre>	

```
./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
```