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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка PNG изображения

Студент гр. 3343 _____ Пименов П.В.

Преподаватель _____ Государкин Я.С.

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент: Пименов Пётр

Группа: 3343

Тема: Обработка PNG изображения

Условия задания (Вариант 5.11):

Программа должна иметь следующую функции по обработке изображений:

1. Разделяет изображение на N*M частей. Флаг для выполнения данной операции: `--split`. Реализация: провести линии заданной толщины. Функционал определяется:
    1. Количество частей по “оси” Y. Флаг `--number_x`. На вход принимает число больше 1.
    2. Количество частей по “оси” X. Флаг `--number_y`. На вход принимает число больше 1.
    3. Толщина линии. Флаг `--thickness`. На вход принимает число больше 0
    4. Цвет линии. 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. Рисование прямоугольника. Флаг для выполнения данной операции: `--rect`. Он определяется:
    1. Координатами левого верхнего угла. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y.
    2. Координатами правого нижнего угла. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y.
    3. Толщиной линий. Флаг `--thickness`. На вход принимает число больше 0.

4. Цветом линий. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет).
5. Прямоугольник может быть залит или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.
6. Цветом, которым он залит, если пользователем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`).

3. Сделать рамку в виде узора. Флаг для выполнения данной операции: `--ornament`. Рамка определяется:
   1. Узором. Флаг `--pattern`. Обязательные значения: rectangle и circle, semicircles.
   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.
4. Поворот изображения (части) на 90/180/270 градусов. Флаг для выполнения данной операции: `--rotate`. Функционал определяется:
   1. Координатами левого верхнего угла области. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y.
   2. Координатами правого нижнего угла области. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y.
   3. Углом поворота. Флаг `--angle`, возможные значения: `90`, `180`, `270`.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 15.05.2024

Дата защиты реферата: 15.05.2024

АННОТАЦИЯ

В ходе курсовой работы реализована программа, осуществляющая обработку PNG изображения. Для взаимодействия с программой реализован интерфейс командной строки (CLI). Программа реализует следующие функции: разделение изображения на N*M частей, рисование прямоугольника, рисование рамки в виде узора, поворот части изображения на 90/180/270 градусов. Сборка проекта осуществляется с помощью утилиты make.

ВВЕДЕНИЕ

Цель работы: понять структуру PNG изображения, научиться работать с PNG изображением на языке программирования C с помощью библиотеки libpng, реализовать программу, реализующую несколько функций по обработке изображения, его считыванию и записи, взаимодействие с которой должно осуществляться с помощью интерфейса командной строки.

1. ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММЫ

Описание структур:

1. *canvas_t* – структура, содержащая двумерный массив пикселей (видимую часть изображения), его ширину и высоту.
2. *image_t* – структура, содержащая данные для работы с PNG изображением: указатель на структуру *canvas_t*, тип изображения (RGB/RGBA/..), глубину цвета и др.
3. *color_t* – структура, которая представляет собой цвет, кодируемый тремя компонентами: *r* (красный), *g* (зеленый), *b* (синий).
4. *px_t* – структура, содержащая координаты пикселя. Используется для реализации алгоритма заливки.
5. *stack_t* – структура, с помощью которой реализована структура данных Стек. Используется в алгоритме заливки.
6. *parameters_t* – структура, в которую заносятся считанные аргументы командной строки.

Описание функций:

1. *int* copy_int(int source)* – динамически выделяет место в оперативной памяти для типа *int*, копирует значение, возвращает указатель.
2. *char* copy_string(char* source)* – динамически выделяет место в оперативной памяти для типа *char**, копирует значение, возвращает указатель.
3. *int parse_int(int** parameter, int* argc)* – считывает аргумент командной строки типа *int*.
4. *int parse_function(int** parameter, int* argc, int function)* – считывает аргумент командной строки типа int – номер функции.
5. *int parse_string(char** parameter, int* argc)* – считывает аргумент командной строки типа *char**.
6. *int parse_color(color_t** parameter, int* argc)* – считывает аргумент командной строки типа *color_t**.

7. *int parse_coordinates(int** parameter_x, int** parameter_y, int* argc)* – считывает координаты из командной строки.
8. *int parse_bool(int** parameter, int* argc)* – считывает бинарное значение из командной строки.
9. *int parse_pattern(int** parameter, int* argc)* – считывает узор для функции рисования рамки из командной строки.
10. *int process(parameters_t* parameters_t)* – выполняет вызов функций обработки изображения, контроль ошибок в процессе выполнения.
11. *int function1(image_t* image, int number_x, int number_y, int thickness, color_t* color)* – вызывает функцию разделения изображения на N*M частей.
12. *int function2(image_t* image, int x0, int y0, int x1, int y1, int thickness, color_t* color, color_t* fill_color)* – вызывает функцию рисования прямоугольника.
13. *int function3(image_t* image, int pattern, color_t* color, int thickness, int count)* – вызывает функцию рисования рамки.
14. *int function4(image_t* image, int x0, int y0, int x1, int y1, int angle)* – вызывает функцию поворота части изображения.
15. *int function99(image_t* image)* – выводит информацию о изображении.
16. *int function100()* – выводит справку об использовании программы.
17. *void free_parameters(parameters_t* parameters)* – очищает память, выделенную для структуры параметров.
18. *int main(int argc, char* argv[])* – главная функция программы, осуществляет обработку аргументов командной строки, вызов функции *process*.
19. *int color_cmp(color_t* a, color_t* b)* – сравнивает два цвета, возвращает 1, если они идентичны, и 0, если нет.
20. *color_t* create_color(png_byte r, png_byte g, png_byte b)* – создает экземпляр структуры *color_t*, возвращает указатель на него.

21. *color_t* create_void_color(color_t* color)* – создает экземпляр структуры *color_t*, который содержит инвертированный исходный цвет.
22. *canvas_t* create_canvas(int width, int height)* – создает экземпляр структуры *canvas_t* – холста размера *width* в ширину, *height* в высоту.
23. *void free_canvas(canvas_t* canvas)* – очищает память, выделенную для экземпляра структуры *canvas_t*.
24. *void push(stack_t* stack, type_t value)* – добавляет значение в Стек.
25. *void pop(stack_t* stack)* – удаляет элемент из Стека.
26. *type_t top(stack_t* stack)* – возвращает последний элемент из Стека.
27. *int is_empty(stack_t* stack)* – возвращает бинарное значение – пустой Стек или нет.
28. *size_t count(stack_t* stack)* – возвращает количество элементов в Стеке.
29. *stack_t* init_stack()* – создает экземпляр структуры *stack_t*, возвращает указатель на него.
30. *void free_stack(stack_t* stack)* – очищает память, выделенную для экземпляра структуры *stack_t*.
31. *int inside(canvas_t* canvas, color_t* color, color_t* border_color, int x, int y)* – возвращает бинарное значение – находится ли данный пиксель внутри области заливки или нет. Используется для алгоритма заливки.
32. *int fill(canvas_t* canvas, color_t* color, color_t* border_color, int x, int y)* – рекурсивный алгоритм заливки.
33. *int fill_canvas(canvas_t* canvas, color_t* color)* – заливает весь холст цветом.
34. *int read_png_file(char* path, image_t* image)* – считывает PNG изображение.
35. *int write_png_file(char* path, image_t* image)* – записывает PNG изображение.
36. *int is_on_canvas(canvas_t* canvas, int x, int y)* – возвращает бинарное значение – находится ли пиксель внутри холста или нет.

37. *int set_pixel(canvas_t* canvas, color_t* color, int x, int y)* – закрашивает пиксель цветом.
38. *color_t* get_pixel(canvas_t* canvas, int x, int y)* – возвращает цвет пикселя.
39. *canvas_t* copy(canvas_t* canvas, int x0, int y0, int x1, int y1)* – копирует область холста в буфер.
40. *int paste(canvas_t* canvas, canvas_t* pasted, color_t* void_color, int x0, int y0)* – накладывает один холст на другой.
41. *int draw_line(canvas_t* canvas, color_t* color, int x0, int y0, int x1, int y1, int thickness)* – рисует прямую линию.
42. *int draw_circumference(canvas_t* canvas, color_t* color, int x0, int y0, int radius, int thickness)* – рисует окружность.
43. *int draw_circle(canvas_t* canvas, color_t* color, int x0, int y0, int radius, int thickness)* – рисует круг.
44. *int draw_rectangle(canvas_t* canvas, color_t* color, color_t* fill_color, int x0, int y0, int x1, int y1, int thickness)* – рисует прямоугольник.
45. *int split(canvas_t* canvas, color_t* color, int number_x, int number_y, int thickness)* – разделяет изображение на N*M частей.
46. *int ornament(canvas_t* canvas, color_t* color, int function, int count, int thickness)* – рисует рамку в виде узора.
47. *int rotate(canvas_t* canvas, int x0, int y0, int x1, int y1, int angle)* – поворачивает область изображения на 90/180/270 градусов.

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

ТЕСТИРОВАНИЕ

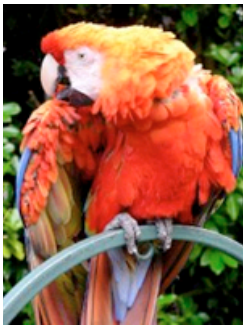

Рисунок 1 – изображение для тестирования

1. Тестирование функции *split*:

   Аргументы для запуска: ./cw --split --number_x 5 --number_y 5 --thickness 10 --color 55.66.77 --input ./test/parrot.png --output ./test/output.png

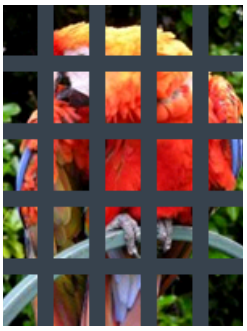

Рисунок 2 – результат работы функции *split*

2. Тестирование функции *rect*:

   Аргументы для запуска: --rect --left_up 30.40 --right_down 100.100 --color 67.94.77 --thickness 10 --fill --fill_color 23.238.75 --input ./test/parrot.png --output ./test/output.png

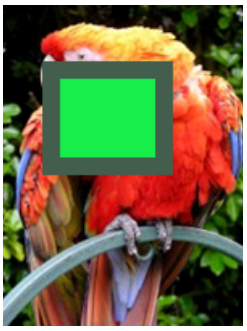

Рисунок 3 – результат работы функции *rect*

3. Тестирование функции *ornament*:

   Аргументы для запуска: ./cw --ornament --pattern circle --color 67.94.77 --count 1 --thickness 10 --input ./test/parrot.png --output ./test/output.png

Рисунок 4 – результат работы функции *ornament*

4. Тестирование функции *rotate*:

   Аргументы для запуска: ./cw --rotate --angle 90 --left_up 30.40 --right_down 100.100 --input ./test/parrot.png --output ./test/output.png

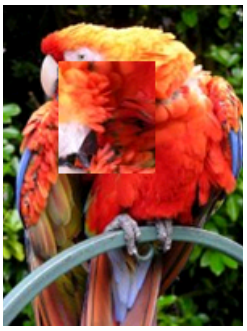

Рисунок 5 – результат работы функции *rotate*

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была создана программа на языке программирования C, осуществляющая обработку PNG изображения. В зависимости от выбранных опций, программа выполняет одну из поддерживаемых функций. Сборка проекта осуществляется с помощью утилиты make. Запуск программы и выбор опций осуществляется через CLI (command line interface).

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: pngdata.h

```
#ifndef PNGDATA_H
#define PNGDATA_H

#include <png.h>

#define BYTES_PER_PIXEL 3

typedef struct {
    int width, height;
    png_bytepp grid;
} canvas_t;

typedef struct {
    int number_of_passes;
    png_byte color_type, bit_depth;
    png_structp png_ptr;
    png_infop info_ptr;
    canvas_t* canvas;
} image_t;

typedef struct {
    png_byte r, g, b;
} color_t;

int color_cmp(color_t* a, color_t* b);
color_t* create_color(png_byte r, png_byte g, png_byte b);
color_t* create_void_color(color_t* color);
canvas_t* create_canvas(int width, int height);
void free_canvas(canvas_t* canvas);

#endif
```

Название файла: pngfill.h

```
#ifndef PNGFILL_H
#define PNGFILL_H

#include "pngdata.h"

int fill_canvas(canvas_t* canvas, color_t* color);
int fill(canvas_t* canvas, color_t* color, color_t* border_color, int x,
int y);

#endif
```

Название файла: pngio.h

```
#ifndef PNGIO_H
#define PNGIO_H
```

```
#include "pngdata.h"

int read_png_file(char* path, image_t* image);
int write_png_file(char* path, image_t* image);

#endif
```

Название файла: pngmnp.h

```
#ifndef PNGMNP_H
#define PNGMNP_H

#include "pngdata.h"

int is_on_canvas(canvas_t* canvas, int x, int y);
int set_pixel(canvas_t* canvas, color_t* color, int x, int y);
color_t* get_pixel(canvas_t* canvas, int x, int y);
canvas_t* copy(canvas_t* canvas, int x0, int y0, int x1, int y1);
int paste(canvas_t* canvas, canvas_t* pasted, color_t* void_color, int
x0, int y0);
int draw_line(canvas_t* canvas, color_t* color, int x0, int y0, int x1,
int y1, int thickness);
int draw_circumference(canvas_t* canvas, color_t* color, int x0, int y0,
int radius, int thickness);
int draw_circle(canvas_t* canvas, color_t* color, int x0, int y0, int
radius, int thickness);
int draw_rectangle(canvas_t* canvas, color_t* color, color_t* fill_color,
int x0, int y0, int x1, int y1,
                   int thickness);
int split(canvas_t* canvas, color_t* color, int number_x, int number_y,
int thickness);
int ornament(canvas_t* canvas, color_t* color, int function, int count,
int thickness);
int rotate(canvas_t* canvas, int x0, int y0, int x1, int y1, int angle);

#endif
```

Название файла: main.c

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

#include "pngdata.h"
#include "pngio.h"
#include "pngmnp.h"

#define COURSE_WORK_INFO "Course work for option 5.11, created by Petr
Pimenov.\n"
#define HELP
\
    "Usage:\n    --split: Split the image into N*M parts.\n
Parameters:\n        --number_x: Number of parts "      \
    "along the X axis.\n        --number_y: Number of parts along the Y
axis.\n        --thickness: Line "             \
    "thickness.\n        --color: Line color.\n        --input (-i):
Input file path.\n        --output (-o): Output " \
```

```
    "file path.\n    Example:\n        ./cw --split --number_x 2 --
number_y 3 --thickness 3 --color 12.34.56 -i "      \
    "input.png -o output.png\n\n    --rect: Draw a rectangle.\n
Parameters:\n        --left_up: Coordinates of "    \
    "the upper left corner.\n        --right_down: Coordinates of the
lower right corner.\n        --thickness: Line " \
    "thickness.\n        --color: Line color.\n        --fill: Is
rectangle filled?\n        "                          \
    "--fill_color: Fill color (if rectangle is filled).\n        --input
(-i): Input file path.\n        --output "    \
    "(-o): Output file path.\n    Example:\n        ./cw --rect --left_up
10.10 --right_down 20.20 --thickness 1 "     \
    "--color 12.34.56 --fill --fill_color 77.88.99 -i input.png -o
output.png\n\n    --ornament: Make a frame "        \
    "in the form of a pattern.\n    Parameters:\n        --pattern:
[rectangle/circles/semicircles] Ornament "         \
    "pattern.\n        --color: Ornament color.\n        --thickness:
Thickness (if needed).\n        --count: Count " \
    "(if needed).\n        --input (-i): Input file path.\n        --
output (-o): Output file path.\n    Example:\n  " \
    "      ./cw --ornament --pattern rectangle --color 12.34.56 --
thickness 10 --count 5 -i input.png -o "             \
    "output.png\n\n    "
\
    "--rotate: Rotate an image part by 90/180/270 degrees\n
Parameters:\n        --left_up: Coordinates of the "    \
    "upper left corner.\n        --right_down: Coordinates of the lower
right corner.\n        --angle: [90/180/270] " \
    "Rotation angle.\n        --input (-i): Input file path.\n        --
output (-o): Output file path.\n    "          \
    "Example:\n        ./cw --rotate --left_up 10.10 --right_down 20.20
--angle 90 -i input.png -o output.png\n\n    " \
    "--info: Print image data.\n    Parameters:\n        --input (-i):
Input file path.\n    Example:\n        ./cw "  \
    "--info -i input.png\n\n    --help (-h): Print usage data.\n
Parameters:\n        No parameters.\n    "         \
    "Example:\n        ./cw --help\n"
#define IMAGE_INFO          "Width: %d\nHeight: %d\nColor type: %d\nBit
depth: %d\nNumber of passes for image interlacing: %d\n"
#define DEFAULT_OUTPUT_FILE "out.png"

typedef struct {
    int argc;
    int* function;
    char* input_path;
    char* output_path;
    int* x0;
    int* y0;
    int* x1;
    int* y1;
    color_t* color;
    int* thickness;
    int* fill;
    color_t* fill_color;
    int* pattern;
    int* count;
    int* angle;
    int* number_x;
```

```
    int* number_y;
} parameters_t;

int* copy_int(int source);
char* copy_string(char* source);
int parse_int(int** parameter, int* argc);
int parse_function(int** parameter, int* argc, int function);
int parse_string(char** parameter, int* argc);
int parse_color(color_t** parameter, int* argc);
int parse_coordinates(int** parameter_x, int** parameter_y, int* argc);
int parse_bool(int** parameter, int* argc);
int parse_pattern(int** parameter, int* argc);
int process(parameters_t* parameters_t);
int function1(image_t* image, int number_x, int number_y, int thickness,
color_t* color);
int function2(image_t* image, int x0, int y0, int x1, int y1, int
thickness, color_t* color, color_t* fill_color);
int function3(image_t* image, int pattern, color_t* color, int thickness,
int count);
int function4(image_t* image, int x0, int y0, int x1, int y1, int angle);
int function99(image_t* image);
int function100();
void free_parameters(parameters_t* parameters);

int
main(int argc, char* argv[]) {
    printf(COURSE_WORK_INFO);

    parameters_t p = {};
    parameters_t* parameters = &p;

    opterr = 0;
    char optstr[] = "hi:o:";
    struct option options[] = {
        {"help", 0, NULL, 'h'},       {"input", 1, NULL, 'i'},
{"output", 1, NULL, 'o'},     {"split", 0, NULL, 300},
        {"number_x", 1, NULL, 301},   {"number_y", 1, NULL, 302},
{"thickness", 1, NULL, 303},  {"color", 1, NULL, 304},
        {"rect", 0, NULL, 305},       {"left_up", 1, NULL, 306},
{"right_down", 1, NULL, 307}, {"fill", 0, NULL, 308},
        {"fill_color", 1, NULL, 309}, {"ornament", 0, NULL, 310},
{"pattern", 1, NULL, 311},    {"count", 1, NULL, 312},
        {"rotate", 0, NULL, 313},     {"angle", 1, NULL, 314},
{"info", 0, NULL, 315},       {NULL, 0, NULL, 0}};

    int getopt_result;
    int success = 1;

    while ((getopt_result = getopt_long(argc, argv, optstr, options,
NULL)) != -1 && success) {
        switch (getopt_result) {
            case 'h': {
                success = parse_function(&parameters->function,
&parameters->argc, 100);
                break;
            }
            case 'i': {
```

```
                success = parse_string(&parameters->input_path,
&parameters->argc);
                break;
            }
            case 'o': {
                success = parse_string(&parameters->output_path,
&parameters->argc);
                break;
            }
            case 300: {
                success = parse_function(&parameters->function,
&parameters->argc, 1);
                break;
            }
            case 301: {
                success = parse_int(&parameters->number_x, &parameters-
>argc);
                break;
            }
            case 302: {
                success = parse_int(&parameters->number_y, &parameters-
>argc);
                break;
            }
            case 303: {
                success = parse_int(&parameters->thickness, &parameters-
>argc);
                break;
            }
            case 304: {
                success = parse_color(&parameters->color, &parameters-
>argc);
                break;
            }
            case 305: {
                success = parse_function(&parameters->function,
&parameters->argc, 2);
                break;
            }
            case 306: {
                success = parse_coordinates(&parameters->x0, &parameters-
>y0, &parameters->argc);
                break;
            }
            case 307: {
                success = parse_coordinates(&parameters->x1, &parameters-
>y1, &parameters->argc);
                break;
            }
            case 308: {
                success = parse_bool(&parameters->fill, &parameters-
>argc);
                break;
            }
            case 309: {
                success = parse_color(&parameters->fill_color,
&parameters->argc);
                break;
```

```
            }
            case 310: {
                success = parse_function(&parameters->function,
&parameters->argc, 3);
                break;
            }
            case 311: {
                success = parse_pattern(&parameters->pattern,
&parameters->argc);
                break;
            }
            case 312: {
                success = parse_int(&parameters->count, &parameters-
>argc);
                break;
            }
            case 313: {
                success = parse_function(&parameters->function,
&parameters->argc, 4);
                break;
            }
            case 314: {
                success = parse_int(&parameters->angle, &parameters-
>argc);
                break;
            }
            case 315: {
                success = parse_function(&parameters->function,
&parameters->argc, 99);
                break;
            }
            case '?':
            default: {
                success = 0;
                printf("Error: Unknown option or missing arguments.\n");
                break;
            };
        }
    }
    if (parameters->input_path == NULL && optind == argc - 1) {
        parameters->input_path = copy_string(argv[argc - 1]);
        parameters->argc += 1;
    }
    int result = 0;
    if (success) {
        result = process(parameters);
    } else {
        result = 40;
    }
    free_parameters(parameters);
    return result;
}

int*
copy_int(int source) {
    int* new_int = (int*)malloc(1 * sizeof(int));
    *new_int = source;
    return new_int;
```

```
}

char*
copy_string(char* source) {
    char* new_string = (char*)calloc(strlen(source) + 1, sizeof(char));
    strncpy(new_string, source, strlen(source) + 1);
    return new_string;
}

int
parse_int(int** parameter, int* argc) {
    if (parameter == NULL || argc == NULL) {
        printf("Error: pointer to parameter or argc is NULL.\n");
        return 0;
    }
    if (*parameter == NULL) {
        int parsed = atoi(optarg);
        if (parsed == 0 && strcmp(optarg, "0") != 0) {
            printf("Error: Could not parse an int.\n");
            return 0;
        }
        *parameter = copy_int(parsed);
        *argc += 1;
    } else {
        printf("Error: Extra option.\n");
        return 0;
    }
    return 1;
}

int
parse_function(int** parameter, int* argc, int function) {
    if (parameter == NULL || argc == NULL) {
        printf("Error: pointer to parameter or argc is NULL.\n");
        return 0;
    }
    if (*parameter == NULL) {
        *parameter = copy_int(function);
        *argc += 1;
    } else {
        printf("Error: Extra option.\n");
        return 0;
    }
    return 1;
}

int
parse_string(char** parameter, int* argc) {
    if (parameter == NULL || argc == NULL) {
        printf("Error: pointer to parameter or argc is NULL.\n");
        return 0;
    }
    if (*parameter == NULL) {
        *parameter = copy_string(optarg);
        *argc += 1;
    } else {
        printf("Error: Extra option.\n");
        return 0;
```

```
    }
    return 1;
}

int
parse_color(color_t** parameter, int* argc) {
    if (parameter == NULL || argc == NULL) {
        printf("Error: pointer to parameter or argc is NULL.\n");
        return 0;
    }
    if (*parameter == NULL) {
        int r = 0, g = 0, b = 0;
        int count = sscanf(optarg, "%d.%d.%d", &r, &g, &b);
        if (count != 3 || r < 0 || r > 255 || g < 0 || g > 255 || b < 0
|| b > 255) {
            printf("Error: Could not parse a color.\n");
            return 0;
        }
        *parameter = create_color((png_byte)r, (png_byte)g, (png_byte)b);
        *argc += 1;
    } else {
        printf("Error: Extra option.\n");
        return 0;
    }
    return 1;
}

int
parse_coordinates(int** parameter_x, int** parameter_y, int* argc) {
    if (parameter_x == NULL || parameter_y == NULL || argc == NULL) {
        printf("Error: pointer to parameter or argc is NULL.\n");
        return 0;
    }
    if (*parameter_x == NULL && *parameter_y == NULL) {
        int x = 0, y = 0;
        int count = sscanf(optarg, "%d.%d", &x, &y);
        if (count != 2) {
            printf("Error: Could not parse coordinates.\n");
            return 0;
        }
        *parameter_x = copy_int(x);
        *parameter_y = copy_int(y);
        *argc += 2;
    } else {
        printf("Error: Extra option.\n");
        return 0;
    }
    return 1;
}

int
parse_bool(int** parameter, int* argc) {
    if (parameter == NULL || argc == NULL) {
        printf("Error: pointer to parameter or argc is NULL.\n");
        return 0;
    }
    if (*parameter == NULL) {
        *parameter = copy_int(1);
```

```
        *argc += 1;
    } else {
        printf("Error: Extra option.\n");
        return 0;
    }
    return 1;
}

int
parse_pattern(int** parameter, int* argc) {
    if (parameter == NULL || argc == NULL) {
        printf("Error: pointer to parameter or argc is NULL.\n");
        return 0;
    }
    if (*parameter == NULL) {
        if (!strcmp(optarg, "rectangle")) {
            *parameter = copy_int(1);
        } else if (!(strcmp(optarg, "circle"))) {
            *parameter = copy_int(2);
        } else if (!(strcmp(optarg, "semicircles"))) {
            *parameter = copy_int(3);
        } else {
            printf("Error: Could not parse an argument.\n");
            return 0;
        }
        *argc += 1;
    } else {
        printf("Error: Extra option.\n");
        return 0;
    }
    return 1;
}

int
process(parameters_t* parameters) {
    int result = 0;
    if (parameters->function == NULL) {
        printf("Error: Function parameter not found.\n");
        return 40;
    }

    int function = *parameters->function;
    if (function == 100) {
        if (parameters->argc == 1) {
            result = function100();
            return result;
        } else {
            printf("Error: Extra option.\n");
            return 40;
        }
    }

    if (parameters->input_path == NULL) {
        printf("Error: Input file path not found.\n");
        return 40;
    }

    image_t i = {};
```

```
    image_t* image = &i;
    result = read_png_file(parameters->input_path, image);
    if (result != 0) {
        return result;
    }

    if (function == 99) {
        if (parameters->argc == 2) {
            result = function99(image);
            free_canvas(image->canvas);
            return result;
        } else {
            printf("Error: Extra option.\n");
            return 40;
        }
    }

    if (parameters->output_path == NULL) {
        parameters->output_path = copy_string(DEFAULT_OUTPUT_FILE);
        parameters->argc += 1;
    }

    if (strcmp(parameters->input_path, parameters->output_path) == 0) {
        printf("Error: Input file is the same as the output file.\n");
        free_canvas(image->canvas);
        return 40;
    }

    if (function == 1) {
        if (parameters->argc == 7 && parameters->number_x && parameters-
>number_y && parameters->thickness
            && parameters->color) {
            result = function1(image, *parameters->number_x, *parameters-
>number_y, *parameters->thickness,
                               parameters->color);
        } else {
            printf("Error: Missing or extra options.\n");
            free_canvas(image->canvas);
            return 40;
        }
    } else if (function == 2) {
        if (!(parameters->x0 && parameters->y0 && parameters->x1 &&
parameters->y1 && parameters->thickness
              && parameters->color)) {
            printf("Error: Missing or extra options.\n");
            free_canvas(image->canvas);
            return 40;
        }
        if (parameters->argc == 9 && !parameters->fill && !parameters-
>fill_color) {
            result = function2(image, *parameters->x0, *parameters->y0,
*parameters->x1, *parameters->y1,
                               *parameters->thickness, parameters->color,
NULL);
        } else if (parameters->argc == 10 && !parameters->fill &&
parameters->fill_color) {
            result = function2(image, *parameters->x0, *parameters->y0,
*parameters->x1, *parameters->y1,
```

```
                                *parameters->thickness, parameters->color,
NULL);
        } else if (parameters->argc == 10 && parameters->fill && !
parameters->fill_color) {
            printf("Error: Missing or extra options.\n");
            free_canvas(image->canvas);
            return 40;
        } else if (parameters->argc == 11 && parameters->fill &&
parameters->fill_color) {
            result = function2(image, *parameters->x0, *parameters->y0,
*parameters->x1, *parameters->y1,
                                *parameters->thickness, parameters->color,
parameters->fill_color);
        } else {
            printf("Error: Missing or extra options.\n");
            free_canvas(image->canvas);
            return 40;
        }
    } else if (function == 3) {
        if (parameters->argc == 7 && parameters->pattern && parameters-
>color && parameters->thickness
            && parameters->count) {
            result = function3(image, *parameters->pattern, parameters-
>color, *parameters->thickness,
                                *parameters->count);
        } else {
            printf("Error: Missing or extra options.\n");
            free_canvas(image->canvas);
            return 40;
        }
    } else if (function == 4) {
        if (parameters->argc == 8 && parameters->x0 && parameters->y0 &&
parameters->x1 && parameters->y1
            && parameters->angle) {
            result = function4(image, *parameters->x0, *parameters->y0,
*parameters->x1, *parameters->y1,
                                *parameters->angle);
        } else {
            printf("Error: Missing or extra options.\n");
            free_canvas(image->canvas);
            return 40;
        }
    }

    result = write_png_file(parameters->output_path, image);
    free_canvas(image->canvas);
    return result;
}

int
function1(image_t* image, int number_x, int number_y, int thickness,
color_t* color) {
    return split(image->canvas, color, number_x, number_y, thickness);
}

int
function2(image_t* image, int x0, int y0, int x1, int y1, int thickness,
color_t* color, color_t* fill_color) {
```

```
    return draw_rectangle(image->canvas, color, fill_color, x0, y0, x1,
y1, thickness);
}

int
function3(image_t* image, int pattern, color_t* color, int thickness, int
count) {
    return ornament(image->canvas, color, pattern, count, thickness);
}

int
function4(image_t* image, int x0, int y0, int x1, int y1, int angle) {
    return rotate(image->canvas, x0, y0, x1, y1, angle);
}

int
function99(image_t* image) {
    printf(IMAGE_INFO, image->canvas->width, image->canvas->height,
image->color_type, image->bit_depth,
           image->number_of_passes);
    return 0;
}

int
function100() {
    printf(HELP);
    return 0;
}

void
free_parameters(parameters_t* parameters) {
    free(parameters->function);
    free(parameters->input_path);
    free(parameters->output_path);
    free(parameters->x0);
    free(parameters->y0);
    free(parameters->x1);
    free(parameters->y1);
    free(parameters->color);
    free(parameters->thickness);
    free(parameters->fill);
    free(parameters->fill_color);
    free(parameters->pattern);
    free(parameters->count);
    free(parameters->angle);
    free(parameters->number_x);
    free(parameters->number_y);
}
```

Название файла: pngfill.c

```
#include <stdlib.h>

#include "pngdata.h"
#include "pngfill.h"
#include "pngmnp.h"

#define STACK_DEFAULT_CAPACITY 10
```

```
typedef struct px_t {
    int x, y;
} px_t;

typedef px_t type_t;

typedef struct stack_t {
    type_t* array;
    int top_index;
    size_t capacity;
} stack_t;

void push(stack_t* stack, type_t value);
void pop(stack_t* stack);
type_t top(stack_t* stack);
int is_empty(stack_t* stack);
size_t count(stack_t* stack);
stack_t* init_stack();
void free_stack(stack_t* stack);

void
push(stack_t* stack, type_t value) {
    if (count(stack) == stack->capacity) {
        stack->capacity += STACK_DEFAULT_CAPACITY;
        stack->array = (type_t*)realloc(stack->array, stack->capacity *
sizeof(type_t));
    }
    stack->array[++(stack->top_index)] = value;
}

void
pop(stack_t* stack) {
    if (is_empty(stack)) {
        return;
    }
    stack->top_index -= 1;
}

type_t
top(stack_t* stack) {
    return stack->array[stack->top_index];
}

int
is_empty(stack_t* stack) {
    return count(stack) == 0;
}

size_t
count(stack_t* stack) {
    return stack->top_index + 1;
}

stack_t*
init_stack() {
    stack_t* stack = (stack_t*)malloc(1 * sizeof(stack_t));
    stack->capacity = STACK_DEFAULT_CAPACITY;
```

```
    stack->array = (type_t*)malloc(stack->capacity * sizeof(type_t));
    stack->top_index = -1;
    return stack;
}

void
free_stack(stack_t* stack) {
    free(stack->array);
    free(stack);
}

int
fill_canvas(canvas_t* canvas, color_t* color) {
    if (canvas == NULL || color == NULL) {
        printf("Error: Null pointer to canvas or color.\n");
        return 42;
    }
    for (size_t row = 0; row < canvas->height; ++row) {
        for (size_t column = 0; column < canvas->width; ++column) {
            set_pixel(canvas, color, column, row);
        }
    }
    return 0;
}

int inside(canvas_t* canvas, color_t* color, color_t* border_color, int
x, int y);

int
inside(canvas_t* canvas, color_t* color, color_t* border_color, int x,
int y) {
    if (canvas == NULL || color == NULL) {
        return 0;
    }
    if (!is_on_canvas(canvas, x, y)) {
        return 0;
    }
    color_t* got_color = get_pixel(canvas, x, y);
    if (border_color != NULL) {
        if (color_cmp(border_color, got_color)) {
            free(got_color);
            return 0;
        }
    }
    int cmp = color_cmp(color, got_color);
    free(got_color);
    if (cmp) {
        return 0;
    }
    return 1;
}

int
fill(canvas_t* canvas, color_t* color, color_t* border_color, int x, int
y) {
    if (canvas == NULL || color == NULL) {
        printf("Error: Null pointer to canvas or color.\n");
        return 42;
```

```
    }
    stack_t* stack = init_stack();
    px_t px = {x, y};
    push(stack, px);
    while (!is_empty(stack)) {
        px_t n = top(stack);
        pop(stack);
        if (inside(canvas, color, border_color, n.x, n.y)) {
            set_pixel(canvas, color, n.x, n.y);
            if (inside(canvas, color, border_color, n.x - 1, n.y)) {
                px_t npx = {n.x - 1, n.y};
                push(stack, npx);
            }
            if (inside(canvas, color, border_color, n.x + 1, n.y)) {
                px_t npx = {n.x + 1, n.y};
                push(stack, npx);
            }
            if (inside(canvas, color, border_color, n.x, n.y - 1)) {
                px_t npx = {n.x, n.y - 1};
                push(stack, npx);
            }
            if (inside(canvas, color, border_color, n.x, n.y + 1)) {
                px_t npx = {n.x, n.y + 1};
                push(stack, npx);
            }
        }
    }
    free_stack(stack);
    return 0;
}
```

Название файла: pngio.c

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

#include "pngdata.h"
#include "pngio.h"

int
read_png_file(char* path, image_t* image) {
    png_byte header[8];
    FILE* fp = fopen(path, "rb");
    if (!fp) {
        printf("Error: Could not open input file.\n");
        return 43;
    }
    fread(header, sizeof(png_byte), 8, fp);
    if (png_sig_cmp((png_const_bytep)header, 0, 8)) {
        printf("Error: Input file is not a PNG file.\n");
        fclose(fp);
        return 43;
    }
    image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (!image->png_ptr) {
        printf("Error: Error with creating PNG read struct.\n");
```

```
        png_destroy_read_struct(&image->png_ptr, &image->info_ptr, NULL);
        fclose(fp);
        return 43;
    }
    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr) {
        printf("Error: Error with creating PNG info struct.\n");
        png_destroy_read_struct(&image->png_ptr, &image->info_ptr, NULL);
        fclose(fp);
        return 43;
    }
    if (setjmp(png_jmpbuf(image->png_ptr))) {
        printf("Error: Unexpected error with reading a PNG file.\n");
        png_destroy_read_struct(&image->png_ptr, &image->info_ptr, NULL);
        fclose(fp);
        return 43;
    }
    png_init_io(image->png_ptr, fp);
    png_set_sig_bytes(image->png_ptr, 8);
    png_read_info(image->png_ptr, image->info_ptr);
    int width = png_get_image_width(image->png_ptr, image->info_ptr);
    int height = png_get_image_height(image->png_ptr, image->info_ptr);
    image->color_type = png_get_color_type(image->png_ptr, image-
>info_ptr);
    if (png_get_color_type(image->png_ptr, image->info_ptr) !=
PNG_COLOR_TYPE_RGB) {
        printf("Error: Input file is not a PNG RGB image.\n");
        png_destroy_read_struct(&image->png_ptr, &image->info_ptr, NULL);
        fclose(fp);
        return 43;
    }
    image->bit_depth = png_get_bit_depth(image->png_ptr, image-
>info_ptr);
    image->number_of_passes = png_set_interlace_handling(image->png_ptr);
    image->canvas = create_canvas(width, height);
    png_read_update_info(image->png_ptr, image->info_ptr);
    png_read_image(image->png_ptr, image->canvas->grid);
    png_destroy_read_struct(&image->png_ptr, &image->info_ptr, NULL);
    fclose(fp);
    return 0;
}

int
write_png_file(char* path, image_t* image) {
    FILE* fp = fopen(path, "wb");
    if (!fp) {
        printf("Error: Could not open output file.\n");
        return 43;
    }
    image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (!image->png_ptr) {
        printf("Error: Error with creating PNG write struct.\n");
        fclose(fp);
        return 43;
    }
    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr) {
```

```
        printf("Error: Error with creating PNG info struct.\n");
        fclose(fp);
        return 43;
    }
    if (setjmp(png_jmpbuf(image->png_ptr))) {
        printf("Error: Unexpected error with writing a PNG file.\n");
        return 43;
    }
    png_init_io(image->png_ptr, fp);
    if (setjmp(png_jmpbuf(image->png_ptr))) {
        printf("Error: Unexpected error after IO session initialization.\
n");
        png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
        fclose(fp);
        return 43;
    }
    png_set_IHDR(image->png_ptr, image->info_ptr, image->canvas->width,
image->canvas->height, image->bit_depth,
                 image->color_type, PNG_INTERLACE_NONE,
PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);
    png_write_info(image->png_ptr, image->info_ptr);
    if (setjmp(png_jmpbuf(image->png_ptr))) {
        printf("Error: Unexpected error after writing a PNG file
metadata.\n");
        png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
        fclose(fp);
        return 43;
    }
    png_write_image(image->png_ptr, image->canvas->grid);
    if (setjmp(png_jmpbuf(image->png_ptr))) {
        printf("Error: Unexpected error after writing a PNG file image
content.\n");
        png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
        fclose(fp);
        return 43;
    }
    png_write_end(image->png_ptr, NULL);
    png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
    fclose(fp);
    return 0;
}
```

Название файла: pngmnp.c

```
#include <stdlib.h>

#include "pngdata.h"
#include "pngfill.h"
#include "pngmnp.h"

#define MIN(x, y) (((x) < (y)) ? (x) : (y))
#define MAX(x, y) (((x) > (y)) ? (x) : (y))

int
is_on_canvas(canvas_t* canvas, int x, int y) {
    if (canvas == NULL) {
        return 0;
    }
```

```
    if (canvas->height <= 0 || canvas->width <= 0) {
        return 0;
    }
    if (x < 0 || x > canvas->width - 1) {
        return 0;
    }
    if (y < 0 || y > canvas->height - 1) {
        return 0;
    }
    return 1;
}

int
set_pixel(canvas_t* canvas, color_t* color, int x, int y) {
    if (canvas == NULL || color == NULL) {
        return 44;
    }
    if (!is_on_canvas(canvas, x, y)) {
        return 44;
    }
    png_bytep row = canvas->grid[y];
    png_bytep pixel = &(row[x * BYTES_PER_PIXEL]);
    pixel[0] = color->r;
    pixel[1] = color->g;
    pixel[2] = color->b;
    return 0;
}

color_t*
get_pixel(canvas_t* canvas, int x, int y) {
    if (canvas == NULL) {
        return NULL;
    }
    if (!is_on_canvas(canvas, x, y)) {
        return NULL;
    }
    png_bytep row = canvas->grid[y];
    png_bytep pixel = &(row[x * 3]);
    return create_color(pixel[0], pixel[1], pixel[2]);
}

canvas_t*
copy(canvas_t* canvas, int x0, int y0, int x1, int y1) {
    if (canvas == NULL) {
        return NULL;
    }
    if (!(0 <= x0 && x0 <= canvas->width && 0 <= y0 && y0 <= canvas-
>height)) {
        return NULL;
    }
    if (!(0 <= x1 && x1 <= canvas->width && 0 <= y1 && y1 <= canvas-
>height)) {
        return NULL;
    }
    canvas_t* new_canvas = create_canvas(x1 - x0, y1 - y0);
    for (int row = y0; row < y1; ++row) {
        for (int column = x0; column < x1; ++column) {
            color_t* got_color = get_pixel(canvas, column, row);
```

```
            if (got_color != NULL) {
                set_pixel(new_canvas, got_color, column - x0, row - y0);
            }
            free(got_color);
        }
    }
    return new_canvas;
}

int
paste(canvas_t* canvas, canvas_t* pasted, color_t* void_color, int x0,
int y0) {
    if (canvas == NULL || pasted == NULL) {
        return 44;
    }
    for (int row = y0; row < pasted->height + y0; ++row) {
        for (int column = x0; column < pasted->width + x0; ++column) {
            color_t* got_color = get_pixel(pasted, column - x0, row -
y0);
            if (void_color != NULL && color_cmp(got_color, void_color)) {
                free(got_color);
                continue;
            }
            set_pixel(canvas, got_color, column, row);
            free(got_color);
        }
    }
    return 0;
}

int
draw_line(canvas_t* canvas, color_t* color, int x0, int y0, int x1, int
y1, int thickness) {
    if (canvas == NULL || color == NULL) {
        return 44;
    }
    if (thickness <= 0) {
        return 44;
    }

    int r = thickness / 2;

    int incline = abs(y1 - y0) - abs(x1 - x0);
    int tmp = 0;
    if (incline > 0) {
        tmp = x0;
        x0 = y0;
        y0 = tmp;
        tmp = x1;
        x1 = y1;
        y1 = tmp;
    }
    if (x0 > x1) {
        tmp = x0;
        x0 = x1;
        x1 = tmp;
        tmp = y0;
        y0 = y1;
```

```
        y1 = tmp;
    }

    int dx = x1 - x0;
    int dy = abs(y1 - y0);
    int error = dx / 2;
    int diry;
    if (y0 < y1) {
        diry = 1;
    } else {
        diry = -1;
    }
    int y = y0;

    for (int x = x0; x <= x1; ++x) {
        int newx, newy;
        if (incline > 0) {
            newx = y;
            newy = x;
        } else {
            newx = x;
            newy = y;
        }
        if (thickness == 1) {
            set_pixel(canvas, color, newx, newy);
        } else {
            draw_circle(canvas, color, newx, newy, r, 1);
        }
        error -= dy;
        if (error < 0) {
            y += diry;
            error += dx;
        }
    }
    return 0;
}

int
draw_circumference(canvas_t* canvas, color_t* color, int x0, int y0, int
radius, int thickness) {
    if (canvas == NULL || color == NULL) {
        return 44;
    }
    if (radius < 0) {
        return 44;
    }
    if (thickness <= 0) {
        return 44;
    }

    int r = thickness / 2;
    int x = radius;
    int y = 0;
    int error = 1 - x;
    while (x >= y) {
        if (thickness == 1) {
            set_pixel(canvas, color, x + x0, y + y0);
            set_pixel(canvas, color, y + x0, x + y0);
```

```
            set_pixel(canvas, color, -x + x0, y + y0);
            set_pixel(canvas, color, -y + x0, x + y0);
            set_pixel(canvas, color, -x + x0, -y + y0);
            set_pixel(canvas, color, -y + x0, -x + y0);
            set_pixel(canvas, color, x + x0, -y + y0);
            set_pixel(canvas, color, y + x0, -x + y0);
        } else {
            draw_circle(canvas, color, x + x0, y + y0, r, 1);
            draw_circle(canvas, color, y + x0, x + y0, r, 1);
            draw_circle(canvas, color, -x + x0, y + y0, r, 1);
            draw_circle(canvas, color, -y + x0, x + y0, r, 1);
            draw_circle(canvas, color, -x + x0, -y + y0, r, 1);
            draw_circle(canvas, color, -y + x0, -x + y0, r, 1);
            draw_circle(canvas, color, x + x0, -y + y0, r, 1);
            draw_circle(canvas, color, y + x0, -x + y0, r, 1);
        }

        y++;
        if (error < 0) {
            error += 2 * y + 1;
        } else {
            x--;
            error += 2 * (y - x + 1);
        }
    }
    return 0;
}

int
draw_circle(canvas_t* canvas, color_t* color, int x0, int y0, int radius,
int thickness) {
    if (canvas == NULL || color == NULL) {
        return 44;
    }
    if (radius < 0) {
        return 44;
    }
    if (thickness <= 0) {
        return 44;
    }

    int r = thickness / 2;
    draw_circumference(canvas, color, x0, y0, radius, thickness);
    for (int y = y0 - r - radius; y < y0 + 1 + r + radius; ++y) {
        for (int x = x0 - r - radius; x < x0 + 1 + r + radius; ++x) {
            if ((x - x0) * (x - x0) + (y - y0) * (y - y0) <= radius *
radius) {
                set_pixel(canvas, color, x, y);
            }
        }
    }
    return 0;
}

int
draw_rectangle(canvas_t* canvas, color_t* color, color_t* fill_color, int
x0, int y0, int x1, int y1, int thickness) {
    if (canvas == NULL || color == NULL) {
```

```
        printf("Error: Null pointer to canvas or color.\n");
        return 44;
    }
    if (thickness <= 0) {
        printf("Error: Thickness is less than 1.\n");
        return 44;
    }

    int r = thickness / 2;

    int lx0 = MIN(x0, x1);
    int ly0 = MIN(y0, y1);
    int lx1 = MAX(x0, x1);
    int ly1 = MAX(y0, y1);

    int dx0 = lx0;
    int dy0 = ly0;
    int dx1 = dx1;
    int dy1 = dy1;
    for (int i = -r; i < 1 + r; ++i) {
        int tx0 = lx0 + i;
        int ty0 = ly0 + i;
        int tx1 = lx1 - i;
        int ty1 = ly1 - i;
        draw_line(canvas, color, tx0, ty0, tx1, ty0, 1);
        draw_line(canvas, color, tx1, ty0, tx1, ty1, 1);
        draw_line(canvas, color, tx0, ty1, tx1, ty1, 1);
        draw_line(canvas, color, tx0, ty0, tx0, ty1, 1);
    }

    if (fill_color != NULL) {
        int nx0 = lx0 + 1 + r;
        int ny0 = ly0 + 1 + r;
        int nx1 = lx1 - 1 - r;
        int ny1 = ly1 - 1 - r;
        while (nx0 <= nx1 && ny0 <= ny1) {
            draw_rectangle(canvas, fill_color, NULL, nx0, ny0, nx1, ny1,
1);
            ++nx0;
            ++ny0;
            --nx1;
            --ny1;
        }
        fill(canvas, fill_color, color, (x0 + x1) / 2, (y0 + y1) / 2);
    }
    return 0;
}

int
split(canvas_t* canvas, color_t* color, int number_x, int number_y, int
thickness) {
    if (canvas == NULL || color == NULL) {
        printf("Error: Null pointer to canvas or color.\n");
        return 44;
    }
    if (number_x <= 1 || number_y <= 1) {
        printf("Error: Number of parts by X or Y axis is less than 2.\
n");
```

```
        return 44;
    }
    if (thickness <= 0) {
        printf("Error: Thickness is less than 1.\n");
        return 44;
    }

    int shift_x = (canvas->width - (number_x - 1) * thickness) /
(number_x);
    int shift_y = (canvas->height - (number_y - 1) * thickness) /
(number_y);
    if (shift_x <= 0) {
        shift_x = 1;
    }
    if (shift_y <= 0) {
        shift_y = 1;
    }

    for (size_t i = 1; i < number_x; ++i) {
        int x = i * shift_x + (i - 1) * thickness;
        for (size_t ik = 0; ik < thickness; ++ik) {
            draw_line(canvas, color, x + ik, 0, x + ik, canvas->height,
1);
        }
    }
    for (size_t j = 1; j < number_y; ++j) {
        int y = j * shift_y + (j - 1) * thickness;
        for (size_t jk = 0; jk < thickness; ++jk) {
            draw_line(canvas, color, 0, y + jk, canvas->width, y + jk,
1);
        }
    }
    return 0;
}

int
ornament(canvas_t* canvas, color_t* color, int function, int count, int
thickness) {
    if (canvas == NULL || color == NULL) {
        printf("Error: Null pointer to canvas or color.\n");
        return 44;
    }
    if (function != 1 && function != 2 && function != 3) {
        printf("Error: Unsupported pattern.\n");
        return 44;
    }
    if (function == 1) { //rectangles
        if (count <= 0 || thickness <= 0) {
            printf("Error: Thickness or count is less than 1.\n");
            return 44;
        }
        for (size_t i = 1; i < count + 1; ++i) {
            int x0 = (i - 1) * 2 * thickness;
            int y0 = x0;
            int x1 = canvas->width - x0 - 1;
            int y1 = canvas->height - y0 - 1;
            if (x0 <= x1 && y0 <= y1) {
                draw_rectangle(canvas, color, NULL, x0, y0, x1, y1, 1);
```

```
                draw_rectangle(canvas, color, NULL, x0 + thickness - 1,
y0 + thickness - 1, x1 - thickness + 1,
                                y1 - thickness + 1, 1);
                fill(canvas, color, NULL, (x0 + x0 + thickness - 1) / 2,
(y0 + y0 + thickness - 1) / 2);
            }
        }
    } else if (function == 2) { //circles
        int width = canvas->width;
        int height = canvas->height;
        canvas_t* new_canvas = create_canvas(width, height);
        fill_canvas(new_canvas, color);
        color_t* void_color = create_void_color(color);

        int radius = MIN(width, height) / 2;
        draw_circle(new_canvas, void_color, width / 2, height / 2,
radius, 1);

        paste(canvas, new_canvas, void_color, 0, 0);
        free(void_color);
        free_canvas(new_canvas);
    } else { //semicircles
        if (count <= 0 || thickness <= 0) {
            printf("Error: Thickness or count is less than 1.\n");
            return 44;
        }
        int r_x = (canvas->width / count) / 2;
        int r_y = (canvas->height / count) / 2;
        if (r_x <= 0) {
            r_x = 10;
        }
        if (r_y <= 0) {
            r_y = 10;
        }
        for (size_t i = 0; i < count; ++i) {
            int x = (2 * i + 1) * (r_x + 1);
            draw_circumference(canvas, color, x, 1, r_x, thickness);
            draw_circumference(canvas, color, x, canvas->height - 1, r_x,
thickness);
        }
        for (size_t j = 0; j < count; ++j) {
            int y = (2 * j + 1) * (r_y + 1);
            draw_circumference(canvas, color, 1, y, r_y, thickness);
            draw_circumference(canvas, color, canvas->width - 1, y, r_y,
thickness);
        }
    }
    return 0;
}

int
rotate(canvas_t* canvas, int x0, int y0, int x1, int y1, int angle) {
    if (canvas == NULL) {
        printf("Error: Null pointer to canvas.\n");
        return 44;
    }
    if (angle != 90 && angle != 180 && angle != 270) {
        printf("Error: Unexpected angle.\n");
```

```
        return 44;
    }
    if (x0 == x1 || y0 == y1) {
        printf("Error: Bad rotation area.\n");
        return 44;
    }
    if (!(0 <= x0 && x0 <= canvas->width && 0 <= y0 && y0 <= canvas-
>height)) {
        printf("Error: Rotation area is not on canvas.\n");
        return 44;
    }
    if (!(0 <= x1 && x1 <= canvas->width && 0 <= y1 && y1 <= canvas-
>height)) {
        printf("Error: Rotation area is not on canvas.\n");
        return 44;
    }

    int cr_x0 = MIN(x0, x1);
    int cr_y0 = MIN(y0, y1);
    int cr_x1 = MAX(x0, x1);
    int cr_y1 = MAX(y0, y1);

    canvas_t* new_canvas = copy(canvas, cr_x0, cr_y0, cr_x1, cr_y1);
    int src_width = new_canvas->width;
    int src_height = new_canvas->height;
    int count = angle / 90;
    for (size_t i = 0; i < count; ++i) {
        int width, height;
        if (i % 2 == 0) {
            width = src_height;
            height = src_width;
        } else {
            width = src_width;
            height = src_height;
        }
        canvas_t* rotated = create_canvas(width, height);
        for (int row = 0; row < new_canvas->height; ++row) {
            for (int column = 0; column < new_canvas->width; ++column) {
                color_t* got_color = get_pixel(new_canvas, column, row);
                set_pixel(rotated, got_color, row, new_canvas->width - 1
- column);
                free(got_color);
            }
        }
        free_canvas(new_canvas);
        new_canvas = rotated;
    }

    int paste_x = (cr_x1 + cr_x0) / 2 - new_canvas->width / 2;
    int paste_y = (cr_y1 + cr_y0) / 2 - new_canvas->height / 2;

    paste(canvas, new_canvas, NULL, paste_x, paste_y);

    free_canvas(new_canvas);
    return 0;
}
```

Название файла: Makefile

```
CC := gcc
CFLAGS :=
BUILDDIR := build
INCLUDEDIR := include
SRCDIR := src
EXECUTABLENAME = cw

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

all: $(EXECUTABLENAME)

$(EXECUTABLENAME): $(OBJECTS)
	$(CC) $(CFLAGS) $(OBJECTS) -lpng -o $@

$(BUILDDIR)/%.o: $(SRCDIR)/%.c
	@mkdir -p $(BUILDDIR)
	$(CC) $(CFLAGS) -I $(INCLUDEDIR) -c $< -o $@

clean:
	@rm -rf $(BUILDDIR) $(EXECUTABLENAME)

.PHONY: all clean
```