**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

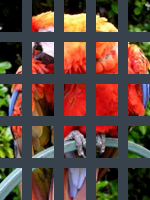


Рисунок 2 – результат работы функции *split*

1. 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

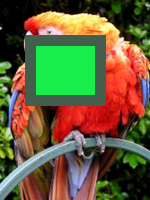


Рисунок 3 – результат работы функции *rect*

1. 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*

1. 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

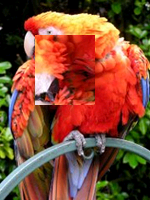


Рисунок 5 – результат работы функции *rotate*

**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