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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка PNG изображения

Студентка гр. 3343 Лобова Е.И.

Преподаватель Государкин Я.С.

Санкт-Петербург
2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студентка: Лобова Екатерина

Группа: 3343

Тема: Обработка PNG изображения

Условия задания (Вариант 4.24):

Программа должна иметь следующую функции по обработке изображений:

- (1) Рисование квадрата с диагоналями. Флаг для выполнения данной операции: `--squared_lines`. Квадрат определяется:
  - Координатами левого верхнего угла. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
  - Размером стороны. Флаг `--side_size`. На вход принимает число больше 0
  - Толщиной линий. Флаг `--thickness`. На вход принимает число больше 0
  - 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)
  - Может быть залит или нет (диагонали располагаются “поверх” заливки). Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.
  - Цветом которым он залит, если пользователем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`)
- (2) Фильтр rgb-компонент. Флаг для выполнения данной операции: `--rgbfilter`. Этот инструмент должен позволять для всего изображения либо установить в диапазоне от 0 до 255 значение заданной компоненты. Функционал определяется

    - Какую компоненту требуется изменить. Флаг `--component_name`. Возможные значения `red`, `green` и `blue`.
    - В какой значение ее требуется изменить. Флаг `--component_value`. Принимает значение в виде числа от 0 до 255
- (3) Поворот изображения (части) на 90/180/270 градусов. Флаг для выполнения данной операции: `--rotate`. Функционал определяется
    - Координатами левого верхнего угла области. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
    - Координатами правого нижнего угла области. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
    - Углом поворота. Флаг `--angle`, возможные значения: `90`, `180`, `270`

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 22.05.2024

Дата защиты реферата: 22.05.2024

АННОТАЦИЯ

В ходе курсовой работы реализована программа, осуществляющая обработку PNG изображения. Для взаимодействия с программой реализован интерфейс командной строки (CLI). Программа реализует следующие функции: рисование квадрата с диагоналями, фильтр rgb-компонент, поворот изображения (части) на 90/180/270 градусов. Сборка проекта осуществляется с помощью утилиты make.

ВВЕДЕНИЕ

Цель работы:

• Изучить структуру изображений в формате PNG.

• Освоить работу с PNG-изображениями на языке программирования C с использованием библиотеки libpng.

• Разработать программу на C, реализующую различные функции обработки изображений, такие как считывание, запись и модификация.

• Обеспечить взаимодействие с программой через интерфейс командной строки для удобства пользователей.

1. ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММЫ

Описание структур:

1. *Png* - структура, содержащая данные для работы с PNG изображением: высоту и ширину изображения, тип цветовой модели изображения, глубина цвета, указатели на структуру управления PNG и структуру информации об изображении, количество проходов, необходимых для полной обработки изображения, указатель на массив указателей на строки пикселей изображения.
2. *Color* – структура, которая представляет собой цвет, задаваемый тремя компонентами: *r* (красный), *g* (зеленый), *b* (синий).
3. *Squared_lines* – структура, содержащая информации об аргументах флагов, связанных с функцией *squared_lines*.
4. *Rgbfilter* – структура, содержащая информации об аргументах флагов, связанных с функцией *rgbfilter*.
5. *Rotate* – структура, содержащая информации об аргументах флагов, связанных с функцией *rotate*.
6. *Configs* – структура, в которую заносятся считанные аргументы командной строки.

Описание функций:

1. *int is_number(const char* str)* – проверяет является ли поданная строка числом. Если да, то возвращает 1, иначе 0.
2. *void raise_error(Configs* config, char* message, int return_code)* – очищает память выделенную в программе динамически, выдаёт сообщение об ошибке и завершает программу.
3. *void init_configs(Configs* config)* – инициализирует структуру *config*.
4. *void free_memory(Configs* config)* – очищает память, выделенную для элементов структуры *config*.
5. *void parsing_coordinates(const char* flag, Configs* config, const char* optarg)* - используется для разбора и проверки координат, указанных в

аргументах командной строки, и для обновления структуры с*onfig* в соответствии с этими координатами.

6. *void parsing_angle(Configs* config, const char* optarg)* - используется для проверки данных на соответствие углу и для обновления структуры *config* в соответствии с этим аргументом.
7. *void parsing_number(const char* flag, Configs* config, const char* optarg)* – используется для проверки аргумента командной строки на соответствие неотрицательному числу и для обновления структуры *config* в соответствии с этим аргументом.
8. *void parsing_color(const char* flag, Configs* config, const char* optarg)* - используется для разбора и проверки цветов, указанных в аргументах командной строки, и для обновления структуры *configs* в соответствии с этими цветами.
9. *void parsing_component_name(Configs* config, const char* optarg)* – используется для проверки аргумента на соответствие названию компоненте цвета и для обновления структуры *configs* в соответствии с этим аргументом.
10. *void parsing_component_value(Configs* config, const char* optarg)* - используется для проверки и обновления значения компонента цвета в структуре *config* на основе аргумента командной строки.
11. *void read_png_file(char *file_name, Png *image)* – считывает PNG изображение.
12. *void write_png_file(char *file_name, struct Png *image)* – записывает PNG изображение.
13. *void rgbfilter(Png *image, char component_name, int component_value)* - устанавливает в диапазоне от 0 до 255 значение заданной компоненты для всего изображения.
14. *void squared_lines(Configs* config, Png* image)* - рисует квадрат с диагоналями.

15. *int rotate(canvas_t* canvas, int x0, int y0, int x1, int y1, int angle)* – поворачивает область изображения на 90/180/270 градусов.
16. *void draw_line(Png* image, int left, int up, int right, int down, int thickness, Color* color, int num_color)* – рисует линию, используя алгоритм Безенхэма.
17. *void set_pixel(Png* image, int y, int x, int num_color, Color* color)* - изменяет цвет отдельных пикселей в изображении PNG.
18. *int RGB_or_RGBA(Png *image)* – возвращает количество байт на пиксель.
19. *void process_file(Configs* config, Png* image)* – выполняет вызов функций обработки изображения.
20. int main(int argc, char **argv) – главная функция программы, осуществляет обработку аргументов командной строки, вызывает функцию *process_file.*

Функции разделены на тематические модули, каждый из которых отвечает за определенную часть функциональности. Такая структура облегчает добавление, изменение или удаление функций в будущем. Кроме того, использование Makefile для сборки программы упрощает управление зависимостями между модулями и автоматизирует процесс компиляции. Разработанный программный код см. в приложении А.

ТЕСТИРОВАНИЕ

Рисунок 1 – изображение для тестирования

1. Тестирование функции *rgbfilter*:

   Аргументы для запуска: ./cw --rgbfilter --component_name red --component_value 45 -i mitsuri.png -o out1.png

Рисунок 2 – результат работы функции *rgbfilter*

2. Тестирование функции *squared_lines*:

   Аргументы для запуска: ./cw --squared_lines --left_up 500.300 --side_size 123 --thickness 5 --color 0.0.0 --fill_color 255.255.255 -i mitsuri.png -o out3.png

Рисунок 3 – результат работы функции *squared_lines*

3. Тестирование функции *rotate*:

   Аргументы для запуска:

   ./cw --rotate --angle 90 --left_up 500.450 --right_down 780.600 -i mitsuri.png -o out2.png

Рисунок 4 – результат работы функции *rotate*

4. Тестирование функции *info*:

   Аргументы для запуска: ./cw -i mitsuri.png --info

   Результат:

   Course work for option 4.24, created by Ekaterina Lobova.

   height: 1337;

   width: 1077;

   color_type: 6;

   bit_depth: 8

ЗАКЛЮЧЕНИЕ

В рамках курсовой работы была разработана программа на языке C для обработки изображений в формате PNG. Программа поддерживает набор функций, которые можно выбрать с помощью командной строки (CLI). Сборка проекта осуществляется с помощью утилиты make, что упрощает управление зависимостями и автоматизирует процесс компиляции.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: image_proccesing.h

```
#ifndef PROCESSING_H
#define PROCESSING_H

#define INFO "height: %d;\nwidth: %d;\ncolor_type: %d;\nbit_depth: %d\n"

#include "main.h"

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

void rgbfilter(Png *image, char component_name, int component_value);
void squared_lines(Configs* config, Png* image);
void rotate(Configs* config, Png* image);
void draw_line(Png* image, int left, int up, int right, int down, int
thickness, Color* color, int num_color);
void set_pixel(Png* image, int y, int x, int num_color, Color* color);
int RGB_or_RGBA(Png *image);
void process_file(Configs* config, Png* image);

#endif
```

Название файла: main.h

```
#ifndef MAIN_H
#define MAIN_H

#include "structs.h"
#include "parsing.h"
#include "png_file_io.h"
#include "image_processing.h"


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

#define HELP \
"--squared_lines - drawing a square with diagonals. Parametrs:\n --
left_up - coordinates of the upper-left corner.the value\
 is set in the format left.up, where left is the x coordinate, up is the
y coordinate\n  --side_size - the size of the side\
  of the square.Accepts a number greater than 0 as input.\n     --
thickness - line thickness.Accepts a number greater than 0 as in\
  put.\n   --color - line color.The color is set by the string
rrr.ggg.bbb, where rrr/ggg/bbb are the numbers specifying the\
 color component. The example --color 255.0.0 sets the color red.\n   --
fill - a binary flag that determines whether the square will\
```

```
 be filled in.\n--fill_color - works similarly to the '--color` flag.\n-
-rgbfilter - allows you to either set the value of a given\
 component for the entire image in the range from 0 to 255. Parametrs:\n
     --component_name - which component needs to be changed.\
 Possible values are red, green and `blue'.\n   --component_value - to
which value the component needs to be changed.Takes\
 a value as a number from 0 to 255.\n--rotate - rotate the image (part)
by 90/180/270 degrees. Parametrs:\n   --left_up - coordi\
 nates of the upper-left corner.the value is set in the format left.up,
where left is the x coordinate, up is the y coordinate\n\
     --left_up - coordinates of the lower-right corner of the area. The
value is set in the format right.down, where right is \
the x coordinate, down is the y coordinate.\n   --angle - angle of
rotation. Possible values: 90, 180, 270.\n"

#define START_SENTENCE "Course work for option 4.24, created by Ekaterina
Lobova.\n"

#define DEFAULT_VALUE "out.png"

#endif
```

Название файла: parsing.h

```
#ifndef PARSING_H
#define PARSING_H

#include "main.h"

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int is_number(const char* str);
void raise_error(Configs* config, char* message, int return_code);
void init_configs(Configs* config);
void free_memory(Configs* config);
void parsing_coordinates(const char* flag, Configs* config, const char*
optarg);
void parsing_angle(Configs* config, const char* optarg);
void parsing_number(const char* flag, Configs* config, const char*
optarg);
void parsing_color(const char* flag, Configs* config, const char* optarg);
void parsing_component_name(Configs* config, const char* optarg);
void parsing_component_value(Configs* config, const char* optarg);

#endif
```

Название файла: png_file_io.h

```
#ifndef PNGIO_H
#define PNGIO_H

#include "main.h"

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

```
void read_png_file(char *file_name, Png *image);
void write_png_file(char *file_name, struct Png *image);

#endif
```

Название файла: structs.h

```
#ifndef STRUCTS_H
#define STRUCTS_H

#include <png.h>

typedef struct Png{
    int width, height;
    png_byte color_type;
    png_byte bit_depth;

    png_structp png_ptr;
    png_infop info_ptr;
    int number_of_passes;
    png_bytep *row_pointers;
}Png;

typedef struct Color{
     size_t r;
     size_t g;
     size_t b;
}Color;

typedef struct Squared_lines{
     int n;
     int coordinates;
     int left, up;
     int size;
     int thickness;
     Color* thickness_color;
     size_t fill;
     Color* fill_color;
}Squared_lines;

typedef struct Rgbfilter{
     int n;
     char component_name;
     size_t component_value;
}Rgbfilter;

typedef struct Rotate{
     int n;
     int coordinates;
     int left, up, right, down;
     size_t angle;
}Rotate;

typedef struct Configs{
     int info;
     char* input;
     Squared_lines* squared_lines;
     Rgbfilter* rgbfilter;
```

```
     Rotate* rotate;
     char* output;
}Configs;

#endif
```

Название файла: image_processing.c

```
#include "../include/image_processing.h"

void process_file(Configs* config, Png* image){
     int flag = 1;
     if (config->input != NULL && config->info == 1){
          read_png_file(config->input, image);
          printf(INFO, image->height, image->width, image->color_type,
image->bit_depth);
          for (int y = 0; y < image->height; y++)
               free(image->row_pointers[y]);
          free(image->row_pointers);
     }
     if (config->input != NULL && config->output != NULL){
          if ((config->rgbfilter)->n == 1){
               if ((config->rgbfilter)->component_name != ' ' &&
(config->rgbfilter)->component_value != 256){
                    read_png_file(config->input, image);
                    rgbfilter(image, (config->rgbfilter)-
>component_name, (config->rgbfilter)->component_value);
                    write_png_file(config->output, image);
               } else flag = 0;
          }
          if ((config->squared_lines)->n == 1){
               if ((config->squared_lines)->coordinates != 0 &&
(config->squared_lines)->thickness_color != NULL && (config-
>squared_lines)->thickness != -1){
                         read_png_file(config->input, image);
                         squared_lines(config, image);
                         write_png_file(config->output, image);
               } else flag = 0;
          }
          if ((config->rotate)->n == 1){
               if ((config->rotate)->coordinates!=0 && (config-
>rotate)->angle!=0){
                    read_png_file(config->input, image);
                    rotate(config, image);
                    write_png_file(config->output, image);
               } else flag = 0;
          }
     }
     if (!flag)
          raise_error(config, "there are not enough flags to set
parameters\n", 42);
     free_memory(config);
}

int RGB_or_RGBA(Png *image){
     if (png_get_color_type(image->png_ptr, image->info_ptr) ==
PNG_COLOR_TYPE_RGB){
          return 3;
```

```
        }
        if (png_get_color_type(image->png_ptr, image->info_ptr) ==
PNG_COLOR_TYPE_RGBA){
           return 4;
        }
}

void rgbfilter(Png *image, char component_name, int component_value) {
    int x,y,color;
    color = RGB_or_RGBA(image);
    for (y = 0; y < image->height; y++) {
        png_byte *row = image->row_pointers[y];
        for (x = 0; x < image->width; x++) {
            png_byte *ptr = &(row[x * color]);
            switch(component_name){
                case 'r':
                    ptr[0] = component_value;
                    break;
                case 'g':
                    ptr[1] = component_value;
                    break;
                case 'b':
                    ptr[2] = component_value;
                    break;
            }
        }
    }
}

void set_pixel(Png* image, int y, int x, int num_color, Color* color){
     if (y < 0 || y >= image->height || x < 0 || x >= image->width)
          return;
     png_byte *row = image->row_pointers[y];
     png_byte *ptr = &(row[x * num_color]);
     ptr[0] = color->r;
     ptr[1] = color->g;
     ptr[2] = color->b;
}

void draw_line(Png* image, int left, int up, int right, int down, int
thickness, Color* color, int num_color) {
    int deltaX = abs(right - left);
    int deltaY = abs(down - up);
    int signX = left < right ? 1 : -1;
    int signY = up < down ? 1 : -1;
    int error = deltaX - deltaY;

    while(1)
   {
     int rectX = left - thickness / 2;
        int rectY = up - thickness / 2;
        for (int y = rectY; y < rectY + thickness; y++) {
           for (int x = rectX; x < rectX + thickness; x++) {
                    set_pixel(image, y, x, num_color, color);
                }
        }
        if (left == right && up == down) break;
        int error2 = error * 2;
```

```
        if(error2 > -deltaY)
        {
            error -= deltaY;
            left += signX;
        }
        if(error2 < deltaX)
        {
            error += deltaX;
            up += signY;
        }
    }
}

void squared_lines(Configs* config, Png* image){
     int num_color = RGB_or_RGBA(image);
     int left = (config->squared_lines)->left;
     int up = (config->squared_lines)->up;
     int size = (config->squared_lines)->size;
     int thickness = (config->squared_lines)->thickness;
     if (thickness % 2==0) thickness--;

     if ((config->squared_lines)->fill_color != NULL){
          for (int y = up; y < up + size - 1; y++){
               for (int x = left; x < left+size-1; x++){
                    set_pixel(image, y, x, num_color, (config-
>squared_lines)->fill_color);
               }
          }
     }

     draw_line(image, left, up, left+size-1, up, thickness, (config-
>squared_lines)->thickness_color, num_color);
    draw_line(image, left, up+size-1, left+size-1, up+size-1, thickness,
(config->squared_lines)->thickness_color, num_color);
     draw_line(image, left, up, left, up+size-1, thickness, (config-
>squared_lines)->thickness_color, num_color);
    draw_line(image, left+size-1, up, left+size-1, up+size-1, thickness,
(config->squared_lines)->thickness_color, num_color);
     draw_line(image, left, up, left+size-1, up+size-1, thickness,
(config->squared_lines)->thickness_color, num_color);
     draw_line(image, left+size-1, up, left, up+size-1, thickness,
(config->squared_lines)->thickness_color, num_color);
}

void rotate(Configs* config, Png* image){
     int num_color = RGB_or_RGBA(image);
     int left = (config->rotate)->left;
     int up = (config->rotate)->up;
     int right = (config->rotate)->right;
     int down = (config->rotate)->down;

     if (left < 0 || left > image->width || up < 0 || up > image->height
|| right < 0 || right > image->width || down < 0 || down > image-
>height){
          raise_error(config, "Rotation area is not on image.\n", 43);
     }

     int height_copy = down - up;
```

```
      int width_copy = right - left;
      //копирование области
      png_bytep *row_pointers_copy =  (png_bytep *)
malloc(sizeof(png_bytep) * height_copy);
      for (int y = 0; y < height_copy; y++)
            row_pointers_copy[y] = (png_byte *) malloc(sizeof(png_byte) *
num_color * width_copy);

         for (int y = up; y < down; y++){
            png_byte *row_copy = image->row_pointers[y];
            for (int x = left; x < right; x++){
                  png_byte *ptr = &(row_copy[x * num_color]);
                  for (int i = 0; i < 3; i++){
                        row_pointers_copy[y - up][(x - left)*num_color + i]
= ptr[i];
                  }
            }
         }
         //поворот скопированной области
         int height_rotate, width_rotate;
         int count = 4 - (config->rotate)->angle / 90;
      for (int i = 1; i<=count; i++){
            height_rotate = width_copy;
            width_rotate = height_copy;

            png_bytep *row_pointers_rotate =  (png_bytep *)
malloc(height_rotate * sizeof(png_bytep));
            for (int y = 0; y < height_rotate; y++)
                  row_pointers_rotate[y] = (png_byte *) malloc(num_color *
width_rotate* sizeof(png_byte));

            for (int y = 0; y < height_copy; y++){
                  png_byte *row_copy = row_pointers_copy[y];
                  for (int x = 0; x < width_copy; x++){
                        png_byte *ptr = &(row_copy[x * num_color]);
                        for (int i = 0; i < 3; i++){
                              row_pointers_rotate[x][(width_rotate - 1 -
y)*num_color + i] = ptr[i];
                        }
                  }
            }
            free(row_pointers_copy);
            row_pointers_copy = row_pointers_rotate;
            width_copy = width_rotate;
            height_copy = height_rotate;

         }
         //вставка на исходное изображение
         int rotate_left = (left+right)/2 - width_rotate/2;
         int rotate_up = (up+down)/2 - height_rotate/2;
      Color color;
      for (int y = rotate_up; y < rotate_up+height_rotate; y++){
            for (int x = rotate_left; x < rotate_left + width_rotate;
x++){
                  color.r = row_pointers_copy[y-rotate_up][(x-
rotate_left)*num_color];
                  color.g = row_pointers_copy[y-rotate_up][(x-
rotate_left)*num_color + 1];
```

```
                color.b = row_pointers_copy[y-rotate_up][(x-
rotate_left)*num_color + 2];
                set_pixel(image, y, x, num_color, &color);
          }
     }
     free(row_pointers_copy);
}
```

Название файла: main.c

```
#include "../include/main.h"

int main(int argc, char **argv) {
     printf(START_SENTENCE);
     Configs config;
     init_configs(&config);
     const char* shortOpts = "i:o:h!";
     const struct option longOpts[] = {
          { "help", no_argument, NULL, 'h' },
          { "input", required_argument, NULL, 'i' },
          { "output", required_argument, NULL, 'o' },
          { "info", no_argument, NULL, '!'},
          {"squared_lines", no_argument, NULL, 0},
          {"rgbfilter",no_argument, NULL, 0},
          {"rotate", no_argument, NULL, 0},
          {"component_name", required_argument, NULL, 0},
          {"component_value", required_argument, NULL, 0},
          {"left_up", required_argument, NULL, 0},
          {"right_down", required_argument, NULL, 0},
          {"angle", required_argument, NULL, 0},
          {"side_size", required_argument, NULL, 0},
          {"thickness", required_argument, NULL, 0},
          {"color", required_argument, NULL, 0},
          {"fill", no_argument, NULL, 0},
          {"fill_color", required_argument, NULL, 0},
          { NULL, 0, NULL, 0 }
     };
     int opt;
     int longIndex;
     struct Png image;
     while((opt = getopt_long(argc, argv, shortOpts, longOpts,
&longIndex)) != -1){
          switch(opt){
               case 'i':
                    if (config.input != NULL){
                         free(config.input);
                    }
                    config.input =
(char*)malloc((strlen(optarg)+1)*sizeof(char));
                    strncpy(config.input, optarg, strlen(optarg));
                    break;
               case 'o':
                    if (config.output != NULL){
                         free(config.output);
                    }
                    config.output =
(char*)malloc((strlen(optarg)+1)*sizeof(char));
                    strncpy(config.output, optarg, strlen(optarg));
                    break;
```

```
                case 'h':
                     printf(HELP);
                     break;
                case '!':
                     config.info = 1;
                     break;
                case '?':
                     printf("found unknown option\n");
                     free_memory(&config);
                     return 0;
                case 0:
                     if (longOpts[longIndex].name == "squared_lines"){
                          (config.squared_lines)->n = 1;
                     }
                     if (longOpts[longIndex].name == "rgbfilter"){
                          (config.rgbfilter)->n = 1;
                     }
                     if (longOpts[longIndex].name == "rotate"){
                          (config.rotate)->n = 1;
                     }
                     if (longOpts[longIndex].name == "component_name"){
                          parsing_component_name(&config, optarg);
                     }
                     if (longOpts[longIndex].name == "component_value"){
                          parsing_component_value(&config, optarg);
                     }
                     if (longOpts[longIndex].name == "left_up"){
                          parsing_coordinates(longOpts[longIndex].name,
&config, optarg);
                     }
                     if (longOpts[longIndex].name == "right_down"){
                          parsing_coordinates(longOpts[longIndex].name,
&config, optarg);
                     }
                     if (longOpts[longIndex].name == "angle"){
                          parsing_angle(&config, optarg);
                     }
                     if (longOpts[longIndex].name == "color" ||
longOpts[longIndex].name == "fill_color"){
                          parsing_color(longOpts[longIndex].name,
&config, optarg);
                     }
                     if (longOpts[longIndex].name == "side_size"){
                          parsing_number(longOpts[longIndex].name,
&config, optarg);
                     }
                     if (longOpts[longIndex].name == "thickness"){
                          parsing_number(longOpts[longIndex].name,
&config, optarg);
                     }
                     if (longOpts[longIndex].name == "fill"){
                          (config.squared_lines)->fill = 1;
                     }
           }
      }
      if (config.input == NULL && optind == argc - 1){
           config.input = (char*)malloc((strlen(argv[argc -
1])+1)*sizeof(char));
```

```
            strncpy(config.input, argv[argc - 1], strlen(argv[argc - 1]));
        }
        if (config.output == NULL){
            config.output =
(char*)malloc((strlen(DEFAULT_VALUE)+1)*sizeof(char));
            strncpy(config.output, DEFAULT_VALUE, strlen(DEFAULT_VALUE));
        }
        process_file(&config, &image);
        return 0;
}
```

Название файла: parsing.c

```
#include "../include/parsing.h"

int is_number(const char* str) {
    int len = strlen(str);
    if (str[0] == '-') len -=1;
    int has_digit = 0;
    while (*str != '\0') {
        if (*str >= '0' && *str <= '9') {
            has_digit+= 1;
        }
        str++;
    }
    return has_digit == len;
}

void raise_error(Configs* config, char* message, int return_code){
        free_memory(config);
        fprintf(stderr, message, return_code);
        exit(return_code);
}

void init_configs(Configs* config){
        config->info = 0;
        config->input = NULL;

        config->squared_lines =
(Squared_lines*)malloc(sizeof(Squared_lines));
        (config->squared_lines)->n = 0;
        (config->squared_lines)->coordinates = 0;
        (config->squared_lines)->left = 0;
        (config->squared_lines)->up = 0;
        (config->squared_lines)->size = -1;
        (config->squared_lines)->thickness = -1;
        (config->squared_lines)->thickness_color = NULL;
        (config->squared_lines)->fill = 0;
        (config->squared_lines)->fill_color = NULL;

        config->rgbfilter = (Rgbfilter*)malloc(sizeof(Rgbfilter));
        (config->rgbfilter)->component_name = ' ';
        (config->rgbfilter)->component_value = 256;

        config->rotate = (Rotate*)malloc(sizeof(Rotate));
        (config->rotate)->coordinates = 0;
        (config->rotate)->left = 0;
        (config->rotate)->up = 0;
        (config->rotate)->right = 0;
```

```
      (config->rotate)->down = 0;
      (config->rotate)->angle = 0;
      config->output = NULL;

}

void free_memory(Configs* config){
      if (config->input != NULL) free(config->input);
      if (config->output != NULL) free(config->output);
      if (config->rgbfilter != NULL) free(config->rgbfilter);
      if (config->squared_lines != NULL){
            if ((config->squared_lines)->thickness_color != NULL)
free((config->squared_lines)->thickness_color);
            if ((config->squared_lines)->fill_color != NULL) free((config-
>squared_lines)->fill_color);
            free(config->squared_lines);
      }
      if (config->rotate != NULL) free(config->rotate);
}

void parsing_coordinates(const char* flag, Configs* config, const char*
optarg){
      int x, y;
      if (sscanf(optarg, "%d.%d", &x, &y) != 2){
            raise_error(config, "incorrect form of agrumeng for flag\n",
41);
      } else {
            if (strcmp(flag, "left_up") == 0){
                  (config->squared_lines)->coordinates = 1;
                  (config->squared_lines)->left = x;
                  (config->squared_lines)->up = y;
                  (config->rotate)->left = x;
                  (config->rotate)->up = y;
            } else {
                  (config->rotate)->coordinates = 1;
                  (config->rotate)->right = x;
                  (config->rotate)->down = y;
            }
      }
}

void parsing_angle(Configs* config, const char* optarg){
      if (strcmp(optarg,"90") == 0 || strcmp(optarg,"180") == 0 ||
strcmp(optarg,"270") == 0){
            (config->rotate)->angle = atoi(optarg);
      } else {
            raise_error(config, "argument for --agngle must be one of 90,
180 or 270\n", 41);
      }
}

void parsing_number(const char* flag, Configs* config, const char*
optarg){
      if (is_number(optarg) && optarg[0]!='-'){
            if (strcmp(flag, "thickness")==0)
                  (config->squared_lines)->thickness = atoi(optarg);
            else (config->squared_lines)->size = atoi(optarg);
```

```
     } else{
          raise_error(config, "incorrect value of the argument\n", 41);
     }
}

void parsing_color(const char* flag, Configs* config, const char*
optarg){
     int  r, g, b;
     if (sscanf(optarg, "%d.%d.%d", &r, &g, &b)!=3){
          raise_error(config, "incorrect form of agrument for flag\n",
41);
     } else {
          if (r >= 0 && r <= 255 && g >= 0 && g <= 255 && b >= 0 && b <=
255){
               if (strcmp(flag, "color") == 0){
                    (config->squared_lines)->thickness_color =
(Color*)malloc(sizeof(Color));
                    ((config->squared_lines)->thickness_color)->r = r;
                    ((config->squared_lines)->thickness_color)->g = g;
                    ((config->squared_lines)->thickness_color)->b = b;
               } else {
                    (config->squared_lines)->fill_color =
(Color*)malloc(sizeof(Color));
                    ((config->squared_lines)->fill_color)->r = r;
                    ((config->squared_lines)->fill_color)->g = g;
                    ((config->squared_lines)->fill_color)->b = b;
               }
          } else {
               raise_error(config, "incorrect value of the color
component\n", 41);
          }
     }
}

void parsing_component_name(Configs* config, const char* optarg){
     if (strcmp(optarg,"red") == 0 || strcmp(optarg ,"blue") == 0 ||
strcmp(optarg,"green") == 0){
          (config->rgbfilter)->component_name = optarg[0];
     }else{
          raise_error(config, "the argument of the component_name flag
must have one of the following values: red, green, or blue\n", 41);
     }
}

void parsing_component_value(Configs* config, const char* optarg){
     if (is_number(optarg)){
          int num = atoi(optarg);
          if (num >= 0 && num <= 255){
               (config->rgbfilter)->component_value = num;
          }else{
               raise_error(config, "incorrect value of the color
component\n", 41);
          }
     }else{
         raise_error(config, "incorrect value of the color component\n",
41);
     }
}
```

Название файла: png_file_io.c

```
#include "../include/png_file_io.h"

void read_png_file(char *file_name, Png *image) {
    int x,y;
    char header[8];

    FILE *fp = fopen(file_name, "rb");
    if (!fp){
        fprintf(stderr, "file could not be opened\n");
        exit(40);
    }

    fread(header, 1, 8, fp);
    if (png_sig_cmp(header, 0, 8)){
     fprintf(stderr, "file is not recognized as a PNG\n");
     fclose(fp);
        exit(40);
    }

    image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (!image->png_ptr){
     fprintf(stderr, "png_create_read_struct failed\n");
     fclose(fp);
        exit(40);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr){
     fprintf(stderr, "png_create_info_struct failed\n");
     png_destroy_read_struct(&image->png_ptr, &image->info_ptr, NULL);
     fclose(fp);
        exit(40);
    }

    if (setjmp(png_jmpbuf(image->png_ptr))){
     fprintf(stderr, "error during init_io\n");
     png_destroy_read_struct(&image->png_ptr, &image->info_ptr, NULL);
     fclose(fp);
        exit(40);
    }

    png_init_io(image->png_ptr, fp);
    png_set_sig_bytes(image->png_ptr, 8);

    png_read_info(image->png_ptr, image->info_ptr);
    image->width = png_get_image_width(image->png_ptr, image->info_ptr);
    image->height = png_get_image_height(image->png_ptr, image->info_ptr);
    image->color_type = png_get_color_type(image->png_ptr, image-
>info_ptr);
    image->bit_depth = png_get_bit_depth(image->png_ptr, image->info_ptr);
    image->number_of_passes = png_set_interlace_handling(image->png_ptr);
    png_read_update_info(image->png_ptr, image->info_ptr);

    if (setjmp(png_jmpbuf(image->png_ptr))){
     fprintf(stderr, "error during read_image\n");
```

```
     fclose(fp);
        exit(40);
    }

    image->row_pointers = (png_bytep *) malloc(sizeof(png_bytep) * image-
>height);
    for (y = 0; y < image->height; y++)
        image->row_pointers[y] = (png_byte *)
malloc(png_get_rowbytes(image->png_ptr, image->info_ptr));

    png_read_image(image->png_ptr, image->row_pointers);

    fclose(fp);
}

void write_png_file(char *file_name, struct Png *image) {
    int x,y;
    FILE *fp = fopen(file_name, "wb");
    if (!fp){
        fprintf(stderr, "file could not be opened\n");
        exit(40);
    }

    image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (!image->png_ptr){
     fprintf(stderr, "png_create_write_struct failed\n");
     fclose(fp);
        exit(40);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr){
     fprintf(stderr, "png_create_info_struct failed\n");
     png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
     fclose(fp);
        exit(40);
    }

    if (setjmp(png_jmpbuf(image->png_ptr))){
     fprintf(stderr, "error during init_io\n");
     png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
     fclose(fp);
        exit(40);
    }

    png_init_io(image->png_ptr, fp);

    if (setjmp(png_jmpbuf(image->png_ptr))){
     fprintf(stderr, "error during writing header\n");
     png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
     fclose(fp);
        exit(40);
    }

    png_set_IHDR(image->png_ptr, image->info_ptr, image->width, image-
>height,
                 image->bit_depth, image->color_type, PNG_INTERLACE_NONE,
```

```
                PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);

    png_write_info(image->png_ptr, image->info_ptr);

    if (setjmp(png_jmpbuf(image->png_ptr))){
     fprintf(stderr, "error during writing bytes\n");
     png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
     fclose(fp);
        exit(40);
    }

    png_write_image(image->png_ptr, image->row_pointers);

    if (setjmp(png_jmpbuf(image->png_ptr))){
        fprintf(stderr, "error during end of write\n");
     fclose(fp);
        exit(40);
    }

    png_write_end(image->png_ptr, NULL);
    png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
    for (y = 0; y < image->height; y++)
        free(image->row_pointers[y]);
    free(image->row_pointers);
    fclose(fp);
}
```

Название файла: Makefile

```
SOURCES = $(wildcard sources/*.c)
OBJECTS = $(patsubst sources/%.c,sources/%.o,$(SOURCES))
CC = gcc
CFLAGS = -lpng

all: cw

cw: $(OBJECTS)
     $(CC) $(OBJECTS) $(CFLAGS) -o cw

sources/%.o : sources/%.c
     $(CC) -c $< -o $@

clean:
 rm -rf sources/*.o cw
```