Университет ИТМО
Факультет Программной инженерии и компьютерной техники

Лабораторная работа №6
по дисциплине “Низкоуровневое программирование”

Выполнил: Кудымов В.С.
Группа: P33012

г. Санкт-Петербург
2020 г

Задание

Повернуть картинку(bmp файл) на заданный угол.

Исходный код

1) bmp_struct.c

```c
#include <stdint.h>
#include <stdio.h>
#include <malloc.h>
#include <math.h>
#include "bmp_struct.h"

void print_error_write(enum write_status status){
    switch (status) {
        case WRITE_OK:
            printf("The file was written successfully\n");
            break;
        case WRITE_ERR:
            printf("Cant write in file\n");
            break;
    }
}

void print_error_read(enum read_status status){
    switch(status){
        case READ_OK:
            printf("The file was read successfully\n");
            break;
        case READ_ERR:
            printf("Read IO error\n");
            break;
        case READ_INVALID_HEADER:
            printf("Header of that bmp file is incorrect\n");
            break;
        case READ_INVALID_BITS:
            printf("Cant read the pixel data. The image is damaged");
            break;
    }
}

struct bmp_header create_bmp_header(struct image *img){
    struct bmp_header header;
    size_t offset= calculate_offset(img -> width);
    header.bfType = 0x4D42;
```

```
    header.bfileSize = ((img -> width + offset)*(img -> height)) +
sizeof(struct bmp_header);
    header.bfReserved = 0;
    header.bOffBits = 54;
    header.biSize = 40;
    header.biPlanes = 1;
    header.biBitCount = 24;
    header.biCompression = 0;
    header.biSizeImage = ((img -> height)*((img -> width)*sizeof(struct
pixel)+offset));
    header.biXPelsPerMeter = 0;
    header.biYPelsPerMeter = 0;
    header.biClrUsed = 0;
    header.biClrImportant = 0;
    header.biWidth = img -> width;
    header.biHeight = img -> height;
    return header;
}
enum write_status write_bmp(FILE* out, struct image *write){
    size_t i;
    uint8_t spare[4] = {0};
    struct bmp_header header;
    size_t offset= calculate_offset(write -> width);

    header = create_bmp_header(write);
    if(!fwrite(&header, sizeof(struct bmp_header), 1, out)) return
WRITE_ERR;
    for(i=0; i < write->height; i++){
        if(!fwrite(&write -> data[write->width*i], sizeof(struct
pixel), write -> width, out)) return WRITE_ERR;
        if(offset != 0) fwrite(spare, sizeof(uint8_t), offset, out);
    }
    return WRITE_OK;
}
uint64_t calculate_offset(uint64_t img_width) {
    uint64_t offset = 4 - (img_width * sizeof(struct pixel)) % 4;
    return offset == 4 ? 0 : offset;
}
enum read_status from_bmp(FILE* in, struct image *read){
    uint8_t spare[4];
    size_t i;
    struct bmp_header* header = malloc(sizeof(struct bmp_header));
```

```
    if (!fread(header, sizeof(struct bmp_header), 1, in)) return
READ_ERR;
    fseek(in, header -> bOffBits, SEEK_SET);
    if(header -> bfType != 0x4D42) return READ_INVALID_HEADER;
    if (header->biBitCount != 24) return READ_INVALID_BITS;
    read->width = header->biWidth;
    read->height = header->biHeight;
    uint64_t offset= calculate_offset(read -> width);
    struct pixel* bitSet = malloc(read->height * read->width *
sizeof(struct pixel));
    for(i=0; i < read -> height; i++){
        if(!fread(bitSet+i*read->width, sizeof(struct pixel),
read->width , in)) return READ_ERR;
        if(offset != 0) fread(spare, offset, 1, in);
    }
    read -> data = bitSet;
    free(header);
    return READ_OK;
}
static int32_t turn_x(double angle, int32_t old_x, int32_t old_y) {
    return round(old_x * cos(angle)+1*old_y*sin (angle));
}
static int32_t turn_y(double angle, int32_t old_x, int32_t old_y) {
    return round(-old_x * sin (angle) + old_y * cos (angle));
}

struct image* rotate(struct image* const source, int32_t parameter){
    struct pixel* bitSet = source->data;
    if(parameter == 90 || parameter == 270 || parameter == -90 ||
parameter == -270){
    uint32_t offset_x = source->width / 2;
    uint32_t offset_y = source->height / 2;
    uint32_t width = source->width;
    uint32_t height = source->height;
    double angle = parameter*M_PI/180;
    size_t i;
    size_t j;
    struct pixel* newSet = malloc(width*height*sizeof(struct pixel));
    for(i=0; i < height; i++){
        for(j=0; j < width; j++){
            int32_t oldX = j - offset_x;
            int32_t oldY = i - offset_y;
```

```
            uint32_t newX = offset_y + round(oldX* cos(angle)+oldY*sin
(angle));
            uint32_t newY = offset_x + round(-oldX* sin (angle) + oldY
* cos (angle));
            *(newSet + newY*height + newX) = *(bitSet + i*width + j);
        }
    }
    source -> width = height;
    source -> height = width;
    source -> data = newSet;
    free(bitSet);
    return source;
    }
    else if(parameter == 180 || parameter == -180){
        uint64_t width = source->width;
        uint64_t height = source->height;
        size_t i;
        size_t j;
        struct pixel* newSet = malloc(width*height*sizeof(struct
pixel));
    for(i=0; i < height; i++){
        for(j=0; j < width; j++){
            *(newSet + (height-i-1)*width + (width-j-1)) = *(bitSet +
i*width + j);
        }
    }
    source -> data = newSet;
    free(bitSet);
    return source;
    } else {
        puts("Only 90, 180, 270. Write in file image without
rotation.");
        return source;
    }
}
```

2) bmp_struct.h

```
#ifndef _BMP_STRUCT_H_
#define _BMP_STRUCT_H_

#include <stdint.h>
#include <stdio.h>
```

```c
struct __attribute__((packed)) bmp_header {
    uint16_t bfType;
    uint32_t bfileSize;
    uint32_t bfReserved;
    uint32_t bOffBits;
    uint32_t biSize;
    uint32_t biWidth;
    uint32_t biHeight;
    uint16_t biPlanes;
    uint16_t biBitCount;
    uint32_t biCompression;
    uint32_t biSizeImage;
    uint32_t biXPelsPerMeter;
    uint32_t biYPelsPerMeter;
    uint32_t biClrUsed;
    uint32_t biClrImportant;
};

struct image {
    uint32_t width, height;
    struct pixel *data;
};

struct __attribute__((packed)) pixel {
    unsigned char b, g, r;
};

enum read_status {
    READ_OK = 0,
    READ_INVALID_BITS,
    READ_INVALID_HEADER,
    READ_ERR
};

enum write_status {
    WRITE_OK = 0,
    WRITE_ERR
};

enum read_status from_bmp(FILE* in, struct image *read);
enum write_status write_bmp(FILE* out, struct image *write);
struct image* rotate(struct image* const in, int32_t parameter);
```

```
struct bmp_header create_bmp_header(struct image *img);
void print_error_read(enum read_status status);
void print_error_write(enum write_status status);
size_t calculate_offset(uint64_t img_width);

#endif
```

3) main.c

```
#include <stdio.h>
#include <math.h>
#include <stdbool.h>
#include "bmp_struct.h"

int main(){
    FILE* in, * out;
    struct image source_img;

    in = fopen("./lights.bmp", "rb");
    if(in == NULL) {
        puts("Can't open file");
        return 0;
    }
    else{
        enum read_status statusRead = from_bmp(in, &source_img);
        rotate(&source_img, 90);
        if(statusRead == READ_OK){
            out = fopen("./red.bmp", "wb");
            if(in == NULL) {
                puts("Can't open file");
                return 0;
            }
            enum write_status statusWrite = write_bmp(out,
&source_img);
            print_error_write(statusWrite);
        } else{
            print_error_read(statusRead);
        }
    }
    fclose(in);
    fclose(out);
    return 0;
}
```

Вывод

Немного научился работать с bmp файлами, а также поворачивать их под различными углами, но им от этого плохо :(.