УНИВЕРСИТЕТ ИТМО

ПИиКТ

Языки Системного Программирования

ЛАБОРАТОРНАЯ РАБОТА №5

Группа: P3200

Студент: Базарова Анна Александровна

2018 год

Санкт-Петербург

Цель: написать на языке С программу, реализующую поворот картинки на 90 градусов вправо или влево. В качестве аргументов программа должна принимать имя файла-изображения в формате BMP любого разрешения.

bmp\_functions.h

#ifndef LOW\_LEVEL\_PRIGRAMMING\_LABS\_BMP\_FUNCTIONS\_H

#define LOW\_LEVEL\_PRIGRAMMING\_LABS\_BMP\_FUNCTIONS\_H

#include <stdint.h>

#include <stdio.h>

/\*\*

\* Тип - файл бмп

\* Это его заголовки

\*/

typedef struct 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;

} \_\_attribute\_\_((packed)) bmp\_header;

/\*\*

\* Тип - пиксель

\* Мы сохраняем его параметры по палитре RGB

\*/

typedef struct pixel\_t{

uint8\_t b,g,r;

} pixel\_t;

/\*\*

\* Тип - картинка

\* Хранятся ширина и высота

\* А также массив из пикселей

\*/

typedef struct image\_t {

uint32\_t width, height;

struct pixel\_t\* data;

} image\_t;

/\*\*

\* Это состояния для считывания

\*/

typedef enum {

READ\_OK,//считывание прошло успешно

READ\_INVALID\_SIGNATURE,//и т.д. копипаста из учебника

READ\_INVALID\_BITS,

READ\_INVALID\_HEADER

} read\_status;

/\*\*

\* Это состояния для записи

\*/

typedef enum {

WRITE\_OK,//успешно

WRITE\_ERROR//догадайтесь сами

} write\_status;

/\*\*

\* Функция для считывания данных из файла - картинки

\* @param in - файл из которого берем данные

\* @param read - наша картинка, но уже как данные

\* @return статус чтения (READ\_OK если все хорошо)

\*/

read\_status from\_bmp(FILE\* in, image\_t\* const read);

/\*\*

\* Функция для записи данных в файл

\* @param out - файл, в который записываем данные

\* @param img - наша картинка, как данные

\* @return статус записи (WRITE\_OK если все хорошо)

\*/

write\_status to\_bmp(FILE\* out, image\_t const\* img );

/\*\*

\* Функция для поворачивания картинки вправо на 90 градусов

\* @param image - исходная картинка

\* @return новая картинка, которая по сути есть повернутая картинка из параметров

\*/

image\_t rotate\_right\_90(image\_t const image);

/\*\*

\* Функция для поворачивания картинки влево на 90 градусов

\* @param image - исходная картинка

\* @return новая картинка, которая по сути есть повернутая картинка из параметров

\*/

image\_t rotate\_left\_90(const image\_t image);

#endif //LOW\_LEVEL\_PRIGRAMMING\_LABS\_BMP\_FUNCTIONS\_H

bmp\_functions.c

#include <stdlib.h>

#include <stdio.h>

#include "bmp\_functions.h"

read\_status from\_bmp(FILE\* in, image\_t\* const read){

bmp\_header header;

fread(&header, sizeof(header), 1, in);//считали заголовки бмп, молодцы

if(header.bfType!=19778){

return READ\_INVALID\_SIGNATURE;

}

if(header.biBitCount !=24){//по заданию глубина цвета должна быть ровно 24

return READ\_INVALID\_BITS;

}

read->width = header.biWidth;//сохраняем только те из заголовков бмп, что нам понадобятся

read->height = header.biHeight;

read->data = malloc(read->width \* read->height \* sizeof(pixel\_t));//выделяем столько памяти, сколько пикселей \* на размер пикселя

int padding = (4 - (read->width \* sizeof(pixel\_t))%4)%4;//это смещение

pixel\_t \*data = read->data;

for (int i = 0; i < read->height; ++i) {

for (int j = 0; j < read->width; ++j) {

fread(data, sizeof(pixel\_t), 1, in);//попиксельно считали картинку, ура

data++;

}

fseek(in, padding, SEEK\_CUR);//"перепрыгиваем" мусор

}

return READ\_OK;

}

write\_status to\_bmp(FILE\* out, image\_t const\* img) {

bmp\_header header;//расставляем все заголовки

header.bfType = 19778;

header.biWidth = abs(img->width);

header.biHeight = img->height;

int padding = (4 - (header.biWidth \* sizeof(pixel\_t)) % 4) % 4;

header.bOffBits = 54;

header.biBitCount = 24;

header.biSizeImage = abs(header.biHeight) \* header.biWidth \* sizeof(pixel\_t) + abs(header.biHeight)\*padding;

header.bfileSize = header.bOffBits + header.biSizeImage;

header.bfReserved = 0;

header.biCompression = 0;

header.biSize = 40;

header.biPlanes = 1;

header.biClrUsed = 0;

header.biClrImportant = 0;

header.biXPelsPerMeter = 2834;

header.biYPelsPerMeter = 2834;

fwrite(&header,sizeof(bmp\_header),1,out);

for (int i = 0; i < abs(header.biHeight); i++) {

for (int j = 0; j < header.biWidth; j++) {

pixel\_t triple;

triple = img->data[header.biWidth\*i + j];//попиксельная запись в файл

fwrite(&triple,sizeof(pixel\_t),1,out);

}

for (int j = 0; j < padding; j++) {

fputc(0x00, out);//заполняем остаток "мусором"

}

}

return WRITE\_OK;

}

image\_t rotate\_left\_90(const image\_t image) {

image\_t \*rotated = malloc(sizeof(image\_t));

rotated->width = image.height;

rotated->height = image.width;

pixel\_t \*rotate = malloc(abs(rotated->width) \* rotated->height \* sizeof(pixel\_t));

for (int i = 0; i < abs(image.height); ++i) {

for (int j = 0; j < image.width; ++j) {

pixel\_t temp = image.data[image.width \* i + j];

rotate[abs(rotated->width) \* (j + 1) - i - 1] = temp;

}

}

rotated->data = rotate;

return \*rotated;

}

image\_t rotate\_right\_90(const image\_t image) {

image\_t \*rotated = malloc(sizeof(image\_t));

rotated->width = image.height;

rotated->height = image.width;

pixel\_t \*rotate = malloc(abs(rotated->width) \* rotated->height \* sizeof(pixel\_t));

for (int i = 0; i < abs(image.height); ++i) {

for (int j = 0; j < image.width; ++j) {

pixel\_t temp = image.data[image.width \* i + j];

rotate[abs(rotated->width) \* (rotated->height -j -1) + i ] = temp;

}

}

rotated->data = rotate;

return \*rotated;

}

main.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <string.h>

#include "bmp\_functions.h"

int main(int argc, char \*argv[]){

//будем передавать в аргументах командной строки файл откуда, затем файл куда, затем right или left

FILE \*file = fopen(argv[1], "rb");//откуда

FILE \*outptr = fopen(argv[2], "wb");//куда

if(!file){

printf("Ошибка! Файл не найден, проверьте правильность введенных аргументов\n");

return 2;

}

image\_t \*image = malloc(sizeof(image\_t));//выделяем память под новую картинку

read\_status result = from\_bmp(file, image); //в файле с бмп функциями

switch (result){

case READ\_INVALID\_HEADER: {

printf("Ошибка! Косяки в заголовках!\n");

return 2;

}

case READ\_INVALID\_BITS: {

printf("Ошибка! Проблемы с глубиной цвета картинки\n");

return 2;

}

case READ\_INVALID\_SIGNATURE:{

printf("Ошибка! Проблемы с сигнатурой!\n");

return 2;

}

case READ\_OK:{

printf("Считывание произошло успешно!\n");

break;

}

}

image\_t rotated;//это наша новая картиночка

if(strcmp(argv[3], "left") == 0){

rotated = rotate\_left\_90(\*image);//в файле с бмп функциями

printf("Совершаем поворот налево...\n");

} else if(strcmp(argv[3], "right") == 0){

printf("Совершаем поворот направо...\n");

rotated = rotate\_right\_90(\*image);//в файле с бмп функциями

} else{

printf("Ошибка! Вы не выбрали напрваление поворота!\n");

return 2;

}

write\_status result\_write = to\_bmp(outptr,&rotated);//в файле с бмп функциями

if(result\_write != WRITE\_OK){

printf("Ошибка с записью в файл!\n");

} else printf("Запись в файл произошла успешно.\n");

free(rotated.data);

}