## Университет ИТМО

## Кафедра ВТ

Лабораторная работа №6

Низкоуровневое программирование

Выполнил: Федоров Сергей

Группа: P33113

Санкт-Петербург

2020 г.

## Задание лабораторной работы:

Реализовать на языке С считывание и запись BMP картинки, а так же функцию поворота этого изображения на произвольный угол.

**Выполнение:**

Lab5/

.

├── Makefile

├── bmp.c

├── bmp.h

├── image.c

├── image.h

├── images

│   ├── lab6\_result.bmp

│   ├── lab6\_test\_big.bmp

│   └── lab6\_test\_smoll.bmp

└── main.c

**main.c**

*//  
// Created by Sergey Fedorov on 11/6/20.  
//*#include **"bmp.h"**#include **<stdio.h>**#define PATH\_TO\_IMAGE **"images/lab6\_test\_big.bmp"**#define PATH\_TO\_IMAGE\_RES **"images/lab6\_result.bmp"**#define ANGLE 45  
  
**int** main() {  
 **struct** bmp\_image\* b\_image;  
 FILE\* test;  
 test = fopen(PATH\_TO\_IMAGE, **"rb"**);  
  
 bmp\_image\_read(&b\_image, test);  
  
 **struct** image\* image;  
  
 bmp\_to\_image(b\_image, &image, 0);  
  
 printf(**"SIZE: %d %d\n"**, image->height, image->width);  
  
 image\_rotate(image, ANGLE \* M\_PI / 180.0);  
  
 **struct** bmp\_image\* b\_image\_2;  
  
 image\_to\_bmp(image, &b\_image\_2, 1);  
  
 printf(**"TOP HEADER: %s %d %d %d\n"**,  
 b\_image->header.bfType, b\_image->header.bfSize, b\_image->header.bfReserved, b\_image->header.bfOffBits);  
  
 printf(**"TOP HEADER: %s %d %d %d\n"**,  
 b\_image\_2->header.bfType, b\_image\_2->header.bfSize, b\_image\_2->header.bfReserved, b\_image\_2->header.bfOffBits);  
  
  
 printf(**"BOTTOM HEADER: %d %d %d %d %d %d %d %d %d %d %d\n"**,  
 b\_image->header.biSize,  
 b\_image->header.biWidth,  
 b\_image->header.biHeight,  
 b\_image->header.biPlanes,  
 b\_image->header.biBitCount,  
 b\_image->header.biCompression,  
 b\_image->header.biSizeImage,  
 b\_image->header.biXPelsPerMeter,  
 b\_image->header.biYPelsPerMeter,  
 b\_image->header.biClrUsed,  
 b\_image->header.biClrImportant  
 );  
  
 printf(**"BOTTOM HEADER: %d %d %d %d %d %d %d %d %d %d %d\n"**,  
 b\_image\_2->header.biSize,  
 b\_image\_2->header.biWidth,  
 b\_image\_2->header.biHeight,  
 b\_image\_2->header.biPlanes,  
 b\_image\_2->header.biBitCount,  
 b\_image\_2->header.biCompression,  
 b\_image\_2->header.biSizeImage,  
 b\_image\_2->header.biXPelsPerMeter,  
 b\_image\_2->header.biYPelsPerMeter,  
 b\_image\_2->header.biClrUsed,  
 b\_image\_2->header.biClrImportant  
 );  
  
 FILE\* wtest;  
 wtest = fopen(PATH\_TO\_IMAGE\_RES, **"wb"**);  
  
 bmp\_image\_write(b\_image\_2, wtest);  
}

**image.h**

*//  
// Created by Sergey Fedorov on 11/6/20.  
//*#include **<stdint.h>**#ifndef LOW\_LEVEL\_PROGRAMMING\_ITMO\_2020\_IMAGE\_H  
#define LOW\_LEVEL\_PROGRAMMING\_ITMO\_2020\_IMAGE\_H  
  
#define M\_PI (3.14159265358979323846)  
  
**struct** pixel {  
 uint8\_t b, g, r;  
};  
  
**struct** image {  
 int32\_t width, height;  
 **struct** pixel\* data;  
};  
  
**void** free\_image(**struct** image\* image);  
  
**void** image\_rotate(**struct** image\* image, **double** angle);  
  
*//* ***TODO****//void image\_blur(struct image\* image);  
//void image\_dilate(struct image\* image);  
//void image\_erode(struct image\* image);*#endif *//LOW\_LEVEL\_PROGRAMMING\_ITMO\_2020\_IMAGE\_H*

**image.c**

*//  
// Created by Sergey Fedorov on 11/6/20.  
//*#include **"image.h"**#include **<float.h>**#include **<math.h>**#include **<stdlib.h>**#include **<stdio.h>  
  
void** free\_image(**struct** image\* image) {  
 **if** (image != NULL) {  
 free(image->data);  
 free(image);  
 }  
  
}  
  
**int** abs\_to\_int(**double** a){  
 **return** abs((**int**) a);  
}  
  
**double** min(**double** a, **double** b) {  
 **return** a < b ? a : b;  
}  
  
**double** max(**double** a, **double** b) {  
 **return** a > b ? a : b;  
}  
  
**double** sind(**double** angle) {  
 **double** angleradians = angle \* M\_PI / 180.0;  
 **return** sin(angleradians) \* M\_PI / 180.0;  
}  
  
**double** cosd(**double** angle) {  
 **double** angleradians = angle \* M\_PI / 180.0;  
 **return** cos(angleradians) \* M\_PI / 180.0;  
}  
  
**struct** positioned\_pixel{  
 **double** x, y;  
 **struct** pixel pixel;  
};  
  
**void** image\_rotate(**struct** image\* image, **double** angle) {  
 **double** center\_x, center\_y, alpha,  
 min\_x = DBL\_MAX,min\_y = DBL\_MAX,  
 max\_x = -DBL\_MAX, max\_y = -DBL\_MAX;  
  
 uint32\_t x, y, i, j, k, base\_x, base\_y, count;  
  
 count = image->width \* image->height;  
  
  
 **struct** positioned\_pixel\* pixels;  
 pixels = malloc(**sizeof**(**struct** positioned\_pixel) \* count);  
  
 center\_x = ((**double**) image->width) / 2;  
 center\_y = ((**double**) image->height) / 2;  
  
 **for** (y = 0, i = 0; y < image->height; y++) {  
 **for** (x = 0; x < image->width; x++, i++) {  
  
 *// Rotation matrix* pixels[i].x = center\_x + (x - center\_x) \* cos(angle) - (y - center\_y) \* sin(angle);  
 pixels[i].y = center\_y + (x - center\_x) \* sin(angle) + (y - center\_y) \* cos(angle);  
 pixels[i].pixel = image->data[i];  
  
 min\_x = min(min\_x, pixels[i].x);  
 min\_y = min(min\_y, pixels[i].y);  
  
 max\_x = max(max\_x, pixels[i].x);  
 max\_y = max(max\_y, pixels[i].y);  
 }  
 }  
  
 image->width = ceil(max\_x - min\_x + 1);  
 image->height = ceil(max\_y - min\_y + 1);  
 free(image->data);  
  
 *// calloc -> make background black* image->data = calloc(image->width \* image->height, **sizeof**(**struct** pixel));  
  
 **for** (i = 0; i < count; ++i) {  
 pixels[i].x -= min\_x;  
 pixels[i].y -= min\_y;  
  
 base\_x = ceil(pixels[i].x);  
 base\_y = ceil(pixels[i].y);  
  
 *// Mix colours by deviation  
 // Positions:  
 // top (1)  
 // ^^^^^  
 // left(4) <- center (0) -> right(2)  
 // vvvvv  
 // bottom (3)* **for** (k = 0; k < 5; ++k) {  
 **switch** (k) {  
 **case** 0:  
 x = base\_x;  
 y = base\_y;  
 alpha = (1 - abs\_to\_int(pixels[i].x - x)) \* (1 - abs\_to\_int(pixels[i].y - y));  
 **break**;  
  
 **case** 1:  
 x = base\_x;  
 y = base\_y - 1;  
 alpha = (1 - abs\_to\_int(x - pixels[i].x)) \* (1 - min(pixels[i].y - y, 1));  
 **break**;  
  
 **case** 2:  
 x = base\_x + 1;  
 y = base\_y;  
 alpha = (1 - min(x - pixels[i].x, 1)) \* (1 - abs\_to\_int(y - pixels[i].y));  
 **break**;  
  
 **case** 3:  
 x = base\_x;  
 y = base\_y + 1;  
 alpha = (1 - abs\_to\_int( x - pixels[i].x)) \* (1 - min(y - pixels[i].y, 1));  
 **break**;  
  
 **case** 4:  
 x = base\_x - 1;  
 y = base\_y;  
 alpha = (1 - min(pixels[i].x - x, 1)) \* (1 - abs\_to\_int(y - pixels[i].y));  
 **break**;  
 }  
  
 **if** (x >= 0 && x < image->width && y >= 0 && y < image->height) {  
 j = y \* image->width + x;  
  
 image->data[j].r += (pixels[i].pixel.r - image->data[j].r) \* alpha;  
 image->data[j].g += (pixels[i].pixel.g - image->data[j].g) \* alpha;  
 image->data[j].b += (pixels[i].pixel.b - image->data[j].b) \* alpha;  
 }  
 }  
 }  
  
 free(pixels);  
}

**bmp.h**

*//  
// Created by Sergey Fedorov on 11/6/20.  
//*#include **"image.h"**#include **<stdio.h>**#include **<stdint.h>**#ifndef LOW\_LEVEL\_PROGRAMMING\_ITMO\_2020\_BMP\_H  
#define LOW\_LEVEL\_PROGRAMMING\_ITMO\_2020\_BMP\_H  
  
**struct \_\_attribute\_\_**((packed)) bmp\_header {  
 **char** bfType[2];  
 uint32\_t bfSize;  
 uint32\_t bfReserved;  
 uint32\_t bfOffBits;  
  
 uint32\_t biSize;  
 int32\_t biWidth;  
 int32\_t biHeight;  
 uint16\_t biPlanes;  
 uint16\_t biBitCount;  
 uint32\_t biCompression;  
 uint32\_t biSizeImage;  
 int32\_t biXPelsPerMeter;  
 int32\_t biYPelsPerMeter;  
 uint32\_t biClrUsed;  
 uint32\_t biClrImportant;  
};  
  
**struct** bmp\_pixel;  
  
**struct** bmp\_image {  
 **struct** bmp\_header header;  
 **struct** bmp\_pixel\* pixels;  
};  
  
**enum** bmp\_read\_result {  
 ***READ\_OK*** = 0,  
 ***READ\_INVALID\_SIGNATURE***,  
 ***READ\_INVALID\_BITS***,  
 ***READ\_INVALID\_HEADER***,  
 ***READ\_INVALID\_BMP\_FORMAT***,  
 ***READ\_INVALID\_BAD\_PIXEL***};  
  
**enum** bmp\_write\_result {  
 ***WRITE\_OK*** = 0,  
 ***WRITE\_BAD\_HEADER***,  
 ***WRITE\_BAD\_BODY***,  
 ***WRITE\_BAD\_OFFSET***};  
  
**enum** bmp\_read\_result bmp\_image\_read(**struct** bmp\_image\*\* bmp\_image, FILE\* file);  
**void** free\_bmp\_image(**struct** bmp\_image\* image);  
  
**void** bmp\_to\_image(**struct** bmp\_image\* bmp\_image, **struct** image\*\* new\_image\_p, **int** free\_bmp);  
**void** image\_to\_bmp(**struct** image\* image, **struct** bmp\_image\*\* new\_bmp\_image\_p, **int** free\_image);  
  
**enum** bmp\_write\_result bmp\_image\_write(**const struct** bmp\_image\* bmp\_image, FILE \* file);  
  
#endif *//LOW\_LEVEL\_PROGRAMMING\_ITMO\_2020\_BMP\_H*

**bmp.c**

*//  
// Created by Sergey Fedorov on 11/6/20.  
//*#include **"bmp.h"**#include **"image.h"**#include **<stdio.h>**#include **<stdint.h>**#include **<stdlib.h>**#include **<string.h>**#include **<float.h>**#include **<math.h>  
  
struct** bmp\_pixel {  
 uint8\_t b, g, r;  
};  
  
**enum** bmp\_read\_result bmp\_image\_read(**struct** bmp\_image\*\* bmp\_image, FILE \* file) {  
 **struct** bmp\_image\* image = malloc(**sizeof**(**struct** bmp\_image));  
 int32\_t row, rowOffset;  
  
 size\_t read\_count = fread(&(image->header), **sizeof**(**struct** bmp\_header), 1, file);  
  
 **if** (read\_count < 1) {  
 free(image);  
 **return *READ\_INVALID\_HEADER***;  
 } **else if** (!(image->header.bfType[0] == **'B'** && image->header.bfType[1] == **'M'**)) {  
 free(image);  
 **return *READ\_INVALID\_SIGNATURE***;  
 } **else if** (  
 (image->header.biSizeImage != 0 && (image->header.bfSize != image->header.bfOffBits + image->header.biSizeImage))  
 || (image->header.biCompression != 0)  
 || (image->header.biPlanes != 1)) {  
 free(image);  
 **return *READ\_INVALID\_BMP\_FORMAT***;  
 } **else if** (image->header.biBitCount != 24) {  
 free(image);  
 **return *READ\_INVALID\_BAD\_PIXEL***;  
 }  
  
 image->pixels = malloc(**sizeof**(**struct** bmp\_pixel) \* image->header.biWidth \* image->header.biHeight);  
  
 rowOffset = image->header.biWidth % 4;  
 fseek(file, image->header.bfOffBits, SEEK\_SET);  
 **for** (row = image->header.biHeight - 1; row >= 0; --row) {  
 read\_count = fread(  
 image->pixels + row \* image->header.biWidth,  
 **sizeof**(**struct** bmp\_pixel),  
 image->header.biWidth, file  
 );  
  
 **if** (read\_count < image->header.biWidth) {  
 free\_bmp\_image(image);  
 **return *READ\_INVALID\_BITS***;  
 }  
  
 **if** (fseek(file, rowOffset, SEEK\_CUR) != 0) {  
 free\_bmp\_image(image);  
 **return *READ\_INVALID\_BITS***;  
 }  
 }  
  
 \*bmp\_image = image;  
 **return *READ\_OK***;  
}  
  
**void** free\_bmp\_image(**struct** bmp\_image\* image) {  
 **if** (image != NULL) {  
 free(image->pixels);  
 free(image);  
 }  
}  
  
**void** bmp\_to\_image(**struct** bmp\_image\* bmp\_image, **struct** image\*\* new\_image\_p, **int** free\_bmp) {  
 **struct** image\* new\_image = malloc(**sizeof**(**struct** image));  
  
 new\_image->height = bmp\_image->header.biHeight;  
 new\_image->width = bmp\_image->header.biWidth;  
 new\_image->data = malloc(**sizeof**(**struct** pixel) \* new\_image->height \* new\_image->width);  
  
 int32\_t row, pos, index;  
  
 **for**(row = 0; row < new\_image->height; row++) {  
 **for**(pos = 0; pos < new\_image->width; pos++) {  
 index = row \* new\_image->width + pos;  
 **struct** pixel converted\_pixel = {  
 bmp\_image->pixels[index].b,  
 bmp\_image->pixels[index].g,  
 bmp\_image->pixels[index].r  
 };  
  
 new\_image->data[index] = converted\_pixel;  
 }  
 }  
  
 \*new\_image\_p = new\_image;  
  
 **if** (free\_bmp) {  
 free\_bmp\_image(bmp\_image);  
 }  
}  
  
**void** image\_to\_bmp(**struct** image\* image, **struct** bmp\_image\*\* new\_bmp\_image\_p, **int** free\_prev\_image) {  
 **struct** bmp\_image\* new\_image = malloc(**sizeof**(**struct** bmp\_image));  
  
 new\_image->header.biHeight = image->height;  
 new\_image->header.biWidth = image->width;  
  
 new\_image->header.bfType[0] = **'B'**;  
 new\_image->header.bfType[1] = **'M'**;  
 new\_image->header.bfOffBits = **sizeof**(**struct** bmp\_header);  
  
 new\_image->header.biSize = 40;  
 new\_image->header.biPlanes = 1;  
 new\_image->header.biBitCount = 24;  
 new\_image->header.biCompression = 0;  
  
 new\_image->header.biSizeImage = new\_image->header.biHeight \*  
 (new\_image->header.biWidth \* **sizeof**(**struct** bmp\_pixel) + new\_image->header.biWidth % 4);  
 new\_image->header.bfSize = new\_image->header.bfOffBits + new\_image->header.biSizeImage;  
  
 new\_image->pixels = malloc(**sizeof**(**struct** bmp\_pixel) \* new\_image->header.biWidth \* new\_image->header.biHeight);  
  
 int32\_t row, pos, index;  
 **for**(row = 0; row < new\_image->header.biHeight; row++) {  
 **for**(pos = 0; pos < new\_image->header.biWidth; pos++) {  
 index = row \* new\_image->header.biWidth + pos;  
  
 **struct** bmp\_pixel converted\_pixel = {  
 image->data[index].b,  
 image->data[index].g,  
 image->data[index].r  
 };  
  
 new\_image->pixels[index] = converted\_pixel;  
 }  
 }  
  
 \*new\_bmp\_image\_p = new\_image;  
  
 **if** (free\_prev\_image) {  
 free\_image(image);  
 }  
  
}  
  
  
**enum** bmp\_write\_result bmp\_image\_write(**const struct** bmp\_image\* image, FILE\* file) {  
 **static** uint8\_t offsetBuffer[] = { 0, 0, 0 };  
 int32\_t row, rowOffset;  
  
 **if** (fwrite(&(image->header), **sizeof**(**struct** bmp\_header), 1, file) == 0) {  
 **return *WRITE\_BAD\_HEADER***;  
 }  
  
 rowOffset = image->header.biWidth % 4;  
 **for** (row = image->header.biHeight - 1; row >= 0; --row) {  
 **if** (fwrite(image->pixels + row \* image->header.biWidth,  
 **sizeof**(**struct** bmp\_pixel), image->header.biWidth, file) < image->header.biWidth) {  
 **return *WRITE\_BAD\_BODY***;  
 }  
  
 **if** (fwrite(offsetBuffer, 1, rowOffset, file) < rowOffset) {  
 **return *WRITE\_BAD\_OFFSET***;  
 }  
 }  
  
 **return *WRITE\_OK***;  
}

**Вывод:**

На самом деле не особо много мыслей по поводу данной лабораторной работе, но вот следующая ремарка найдется:

Понравился способ того как мы читаем bmp картинку и то что по факту в бинарном файле у нас записана наша же структура указанная в коде, что, до этого момента, я не видел ни в каком другом языке.

Еще во время выполнения работы, приходилось часто дебажить код из-за неявных кастов типов, что достаточно сильно отражалось на финальном качестве картинки.