Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики.

Кафедра вычислительной техники

Языки системного программирования

Лабораторная работа №5

Поворот BMP файла

Выполнил:

Студент группы P3210

Глушков Дмитрий Сергеевич

Санкт-Петербург
2018 г.

Задание:
You have to create a program to rotate a BMP image of any resolution to 90 degrees clockwise.
1. Describe the pixel structure struct pixel to not work with the raster table directly (as with completely structureless data). This should always be avoided.
2. Separate the inner image representation from the input format. The rotation is performed on the inner image format, which is then serialized back to BMP. There can be changes in BMP format, you might want to support other formats, and you do not want to couple the rotation algorithm tightly to BMP.
To achieve that, define a structure structure image to store the pixel array (continuous, now without
padding) and some information that should really be kept. For example, there is absolutely no need to store
BMP signature here, or any of the never-used header fields. We can get away with the image width and
height in pixels.
You will need to create functions to read an image from BMP file and to write it to BMP file (probably
also to generate a BMP header from the inner representation).
3. Separate file opening from its reading.
4. Make error handling unified and handle errors in exactly one place (for this very program it is enough).

Текст программы:
Файл main.c:
```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include "bmp_lib.h"

int main()
{
      FILE *infile, *outfile;
      image_t *img;
      uint32_t status;
      if ((infile = fopen("input.bmp", "r+b")) == NULL)
      {
            puts ("File not found");
            return 1;
      }
      if ((outfile = fopen("output.bmp", "w+b")) == NULL)
      {
            puts ("Can't create new file");
            return 1;
      }
      img = (image_t*)malloc(sizeof(image_t));
      if ((status = input_bmp(infile, img)) != 0) print_error_in(status);
      turn (img);
  if ((status = output_bmp(outfile, img)) != 0) print_error_out(status);
      puts("Process finished");
```

```
      return 0;
}
```

Файл bmp_turn.c:

```
#include <stdio.h>
#include <stdlib.h>
#include "bmp_lib.h"

void turn(image_t* const img)
{
      size_t w, h;
      pixel_t* m;
      w = img -> width;
      h = img -> height;
      m = malloc(sizeof(pixel_t) * w * h);
      for (int i = 0; i < w; i++)
            for (int j = 0; j < h; j++)
      *(m + i*h + j) = *(img->data + (h  - j - 1)*w  + i);
  free(img -> data);
      img -> width = h; img -> height = w;
      img -> data = m;
}
```

Файл bmp.io:

```
#include <stdio.h>
#include <stdlib.h>
#include "bmp_lib.h"

void print_error_in(uint32_t status)
{
      switch (status)
      {
            case 1: puts ("Error: READ_INVALID_SIGNATURE");
            break;
    case 2: puts ("Error: READ_INVALID_BITS");
            break;
    case 3: puts ("Error: READ_INVALID_SIGNATURE");
            break;
    case 4: puts ("Error: READ_INVALID_RESERVED1");
            break;
    case 5: puts ("Error: READ_INVALID_RESERVED2");
            break;
  }
}

void print_error_out(uint32_t status)
{
      switch (status)
      {
            case 1: puts ("Error: WRITE_INVALID_HEADER");
            break;
    case 2: puts ("Error: WRITE_INVALID_BITS");
            break;
  }
}

read_status_t input_bmp (FILE *infile, image_t *img)
{
  read_status_t status;
  header_t hdr;
```

```
  uint32_t memsize = img -> width * img -> height * 3;
  status = read_header(infile, &hdr);
  if (status) return status;
  img -> width = hdr.biWidth;
  img -> height = hdr.biHeight;
  img -> data = (pixel_t*)malloc((size_t)memsize);
  status = read_pixels(infile, &hdr, img -> data);
  return status;
}

read_status_t read_header (FILE *infile, header_t *hdr)
{
  uint32_t size = fread(hdr, 1, sizeof(header_t), infile);
      if (hdr -> bfType != 0x4D42)
            return READ_INVALID_SIGNATURE;
      if (hdr -> bfReserved1)
            return READ_INVALID_RESERVED1;
  if (hdr -> bfReserved2)
            return READ_INVALID_RESERVED2;
      if (size != sizeof(header_t))
            return READ_INVALID_HEADER;
      return READ_OK;
}

read_status_t read_pixels (FILE *infile, header_t *hdr, pixel_t *data)
{
  uint32_t width = hdr -> biWidth, height = hdr -> biHeight, offset = hdr ->
bfOffBits, bitc = hdr -> biBitCount / 8;
      uint32_t padding;
      if (width * 3 % 4 == 0)
    padding = 0;
  else
    padding = 4 - (width * 3 % 4);
      fseek(infile, offset, SEEK_SET);
      for (uint32_t i = 0; i < height; i++)
      {
            uint32_t size = fread(data + width * i, bitc, width, infile);
            if (size != width)
                  return READ_INVALID_BITS;
            fseek(infile, padding, SEEK_CUR);
      }
      return READ_OK;
}

write_status_t output_bmp (FILE *outfile,  image_t *img)
{
  header_t hdr;
  uint32_t padding;
  pixel_t *outdata = img -> data;
  uint64_t imgsize, filesize;
  write_status_t status;

      if (img -> width * 3 % 4 == 0)
    padding = 0;
  else
    padding = 4 - (img -> width * 3 % 4);
  imgsize = (img -> width * 3 + padding) * img -> height;
  filesize = sizeof(hdr) + imgsize;

  hdr.bfType = 0x4D42;
      hdr.bfileSize = filesize;
```

```
	hdr.bfReserved1 = 0;
	hdr.bfReserved2 = 0;
	hdr.bfOffBits = 54;
	hdr.biSize = 40;
	hdr.biWidth = img -> width;
	hdr.biHeight = img -> height;
	hdr.biPlanes = 1;
	hdr.biBitCount = 24;
	hdr.biCompression = 0;
	hdr.biSizeImage = imgsize;
	hdr.biXPelsPerMeter = 0;
	hdr.biYPelsPerMeter = 0;
	hdr.biClrUsed = 0;
	hdr.biClrImportant = 0;

	if (fwrite(&hdr, 1, sizeof(hdr), outfile) != sizeof(hdr))
		return WRITE_INVALID_HEADER;

	for (uint64_t i = 0; i < img -> height; i++)
	{
		if (fwrite(outdata + img -> width * i, 1, 3 * img -> width, outfile)
!= img -> width * 3)
			return WRITE_INVALID_BITS;
		fseek(outfile, padding, SEEK_CUR);
	}

	return WRITE_OK;
}
```

Вывод:
В результате выполнения лабораторной работы была реализована программа, принимающая на вход изображение формата .bmp, поворачивающая его на 90 градусов по часовой стрелке и записывающая полученный результат в новый файл.