BMP File Manipulation

Abbas Aliyev, Aziz Salimli

May 2022

1 Objective

Our main task is to process images in **bmp** format.

Images in **bmp** format are saved every pixel by expansion, that is, you can access every pixel. And our task is to automatically adjust the image for contrast and adjust the contrast.

The brightness of an image is a measure of how strongly a color appears from black. Brightness ranges from 0 to 255. The higher this value, the brighter the pixel. And in order to change the brightness of the image, we must raise the brightness of each pixel.

Contrast is the difference in brightness between different parts of an image. Contrast also takes values from 0 to 255. For example, if an image has a spread of 130 to 160, this image is considered to be an image with poor contrast.

What should we do? Let's say we have image $\boldsymbol{A}$ with spread from $\forall$ x to $\forall$ y$\rightarrow \boldsymbol{A} = [x, y \mid x, y \in A, x \neq 0, x \neq y, x < y]$, we must subtract the arrival of x from everyone. Scatter identification: $[0, x - y]$. After that we have to $255/(x - y)$. And the resulting answer is multiplied by each element in $[0, x - y]$. After this is done, the pixel values of the image will range from 0 to 255.

Due to the fact that in reality the picture consists of three Red Green Blue(RGB) colors, we must do the above manipulations with each of the colors in the image.

2 Code Explanation

2.1 bmp_core.c and bmp_core.h

We have 4 main structures BMP_HEADER DIB_HEADER IMAGE_DATA BMP_FILE. Later we will analyze them in more detail. But first, I want to note the structures that we declare, but do not create in our project (because we believe that they are not specified in our Technical Task).

2.1.1 Declare, but not used

If less than 8 bits are specified for the color, then we will be forced to use COLOR_TABLE, but it is not needed in our task.
BIT_MASK is defined data that is used for masking - selecting individual bits or fields from multiple bits from a binary string or number.
COLOR_PROFILE is a set of data that characterizes a color input or output device or color space according to standards promulgated by the International Color Consortium.

```
typedef char BIT_MASKS;
typedef char COLOR_TABLE;
typedef char COLOR_PROFILE;
```

2.1.2 BMP_HEADER

BMP_HEADER - It stores some information about our file, specifically the first 2 bytes, then 4 bytes, the next 4 bytes are reserved for the application and the last 4 bytes indicate image offsets.

```
struct BMP_HEADER_;
```

2.1.3 DIB_HEADER

DIB_HEADER - stores information about 4 bytes header size, 4 bytes for image width, 4 for height, 2 for colour planes, 2 for pixels, and the remaining 24 are not important.

```
struct DIB_HEADER_;
```

2.1.4 IMAGE_DATA

IMAGE_DATA is a structure that stores the height and width of the image.

```
struct IMAGE_DATA_;
```

2.1.5 BMP_FILE

BMP_FILE - a structure that stores all the structures, the root, the head of our project.

```
struct BMP_FILE;
```

2.1.6 *_read_bmp_file

This function takes the open file handler that we open in main, and then we pass the file to the following functions: read_DIP_HEADER, read_IMAGE_DATA, read_BMP_HEADER.

```
BMP_FILE * read_bmp_file(FILE *file, bool is_big_endian);
```

2.1.7 read_BMP_HEADER, read_DIP_HEADER

Each of these function read a constant-sized header and simply return what they read. Size BMP is 14, when DIP is 56 bytes.

```
static BMP_FILE * read_BMP_HEADER(FILE *f);
static BMP_FILE * read_DIB_HEADER(FILE *f);
```

2.1.8 *_read_IMAGE_DATA

This function can read pixels stored in both, little and big endian. Their order may be RGBA or ABGR. And each of these methods must be read separately. Here we consider the pixels themselves with which we manipulate.

```
static IMAGE_DATA * _read_IMAGE_DATA(FILE *f, uint32_t offset, uint32_t width, uint32_t heightbool is_big_endian);
```

2.1.9 *copy_bmp_structure

The function creates a copy of the bmp structure.

```
BMP_FILE * copy_bmp_structure(const BMP_FILE *input);
```

2.1.10 free_bmp_structure

Since we are working with dynamic memory, we must allocate memory and deallocate memory. This is the function that releases it.

```
void free_bmp_structure(BMP_FILE *bmpFile);
```

2.2 auto_brightness.c

In this file, we have only one function that implements the logic described above. We apply a rule to increase the contrast and brightness of the image.

```
BMP_FILE * auto_adjust_image(BMP_FILE *input);
```

2.3 main.c

In main, we process the input arguments. We also do detection, processing, input and output redirection. We also have help for the user, a brief description of the use of the program.

3 Specification

Our source codes should be compiled with C23 standard. We do not guaranty anything if compiled with any other standard.
Our project only works with 32-bit color .bmp files. We do not guaranty anything if ran with any other file.

4 User Manual

Our program is designed to equalize the contrast and brightness in any picture. The only thing you need is to put the picture in a folder and specify its correct name.

4.1 Installation

To install our program you should follow these simple steps:

- Download files from download zip.
- Make sure you have the CMAKE compiler installed on your computer.
- Run following commands to build the project:
  1. `cd auto_brightness_project`
  2. `mkdir build`
  3. `cd build`
  4. `cmake ../`
  5. `cmake –build .`
  6. `mv auto_brightness_project ../`

4.2 Execution

To run our code you should get back to working directory and run the executable. To run it properly you must provide input file or redirect input.

- Direct execution: `auto_brightness_project image.bmp -o result.bmp`
- Implicit output: `auto_brightness_project image.bmp`
- I/O redirection: `auto_brightness_project < image.bmp > result.bmp`
- Defining endiannes: `auto_brightness_project image.bmp -e l -o result.bmp`

You can mix execution options with each other:

- Direct input with output redireciton: `auto_brightness_project image.bmp > result.bmp`
- Implicit output with input redirection: `auto_brightness_project < image.bmp`
  in this case we will have result with name `stdin.xxxxxxxx.bmp`

5 Example

Here is the picture we had:

And here's what we got!