UFAZ DSA C Project

Manipulating Bitmap (BMP) images

You can download an example BMP image there: `https://tinyurl.com/yaj99chn`. It is a white image with a black strip in its middle.

BMP images store pixels, but because they can use different colours or formats, the file starts with a "header" that describes how pixels are stored. You can find a description of the BMP format and its headers there: `https://en.wikipedia.org/wiki/BMP_file_format`

The first header of a BMP image is as follows (important information in **bold face**):

- 2 first Bytes: 42 4D which are the ASCII values for B M (cf. BMP format)
- **4 next Bytes: the size of the file** 48 C0 12 00 in the `example.bmp` file. NB: this is "little endian"[1]: as you saw in Computer Architecture you need to inverse the bytes to convert them to an unsigned integer, So the value is 00 12 C0 48, which corresponds to 1 228 872 in decimal.
- 4 next Bytes : reserved for application. Not used.
- 4 Bytes : the offset which tells at which byte the image starts. In the `example.bmp` file : 00 00 00 46 = 70 (this is the size of both BMP and DIB headers).

Now comes the second header, the DIB header:

- 4 Bytes in little endian : size of the header starting counting from this point. Here 00 00 00 38h = 56d. Looking in the BMP format, we see this means BITMAPV3INFOHEADER (and because before the DIB, we have 14 Bytes, 14+56 = 70 size of the header before pixels data so all is well.
- **4 Bytes in little endian for image width**: here 80 02 00 00 in little endian, meaning 00 00 02 80 = 640 (the width of the image - you can verify yourself). **You will need to modify this**.
- 4 Bytes in little endian for image height: here E0 01 00 00 in little endian, means 00 00 01 E0 = 480 (you can verify for yourself).
- 2 Bytes : number of color planes (here it should be 1) (this is the case in our example bmp image).
- 2 Bytes : number of bits per pixels : here 20 00 which means 00 20 => 32
- 24 remaining Bytes : not important.

Now should come the colour table but... because we are using 32 bits per pixels, the colour table is not needed :-)

So we now start with pixels at offset 70. We find FF FF FF 00 (this means R V B are FF = white colour, then 0 for alpha channel) 270 times, then 00 00 00 00 (black) 100 times, then FF FF FF 00 270 times = 640 pixels for one line.

Hint1 : If the number of pixels is not multiple of 4, there is padding with the values 0.
Hint2 : Pixels are arranged from left to right BUT from bottom to top!

[1] https://en.wikipedia.org/wiki/Endianness

1 Simple watermarking of BMP images

Nowadays, digital signatures are more and more used to sign documents. This is problematic because anyone can copy a signature from a document to paste it on a new document, so digital signatures are not meaningful anymore.

The aim of this project is to have you create a program that will take a signature stored in BMP format and modify it to "watermark" the signature[2].

The idea is the following. Your program will be called `wm` for "watermark". Executing the following command line:
`$ wm sign.bmp -text "Hello" -date -color FFFFFE -pos 10 20 -o modsign.bmp`
will create a new `modsign.bmp` image where on position 10,20 from the top left of the image, you will write "Hello" followed by the current date (the output of the `date` linux command) *in morse code*, where a dot is a pixel of color `FFFFFE` (nearly white), and a dash is a "line" made of 3 pixels. Space between dots and dashes is one background pixel. Space between letters is 3 pixels, space between 2 words is 5 pixels[3].

If the resolution is high, pixels will be really small, and the line will be quite invisible. Only people who know it is there and where to look for it will find it.

If the line is too long to fit in the image, you need to continue on the next line.

If I watermark the signature by writing the name of the document and the date in morse code in it, and someone makes a digital copy of the signature and uses it on another document, I can later prove that the signature is not the good one because the name of the document and the date will be wrong.

In file `sign.bmp` below:

I signed using a pen with colour #243E7B. Then, I watermarked the file `modsign.bmp` with "wm" written in morse code. The result is:

Can you see/find the watermark with your eyes?

Hint: look for colour #243E7C in the file...

Command line:
`$ wm sign.bmp -text "Hello" -date -color FFFFFE -pos 10,20 -o modsign.bmp`
works the following way:

`sign.bmp` is the name of the file you must modify.

`-text "Hello"` means you must write "hello" in morse code. Note that `-text` is an option. If it is not present, do not write any text.

`-date` you must write in morse code the output of system command `date` after the text (if option `-text` is present. Note that `-date` is an option. If it is not present, do not write the date.

`-pos` Write the morse code at position x,y starting from the top left of the image. If `-pos` is not present write from the top left position of the image (0,0).

[2] `https://en.wikipedia.org/wiki/Digital_watermarking`
[3] http://ascii-table.com/morse-code.php

`-o modsign.bmp` writes the outout (`-o`) in file `modsign.bmp`. If no `-o` option is mentioned, output on the standard output (the user will need to write > `modsign.bmp` to output in `modsign.bmp`.

2 Skills you will acquire with this project

1. Understand what is a file format.
2. Learn to deal with bytes and not integers.
3. Learn about how to use big little-endian values.
4. Learn how to open / write files.

Use a hexadecimal editor to view (and possibly modify) the BMP file to make sure everything is correct.

3 What must be handed back

1. Source codes of your project.
2. A 3 to 5 pages description of your project in LaTeX, which you can have freely access to by opening an account at: `http://www.overleaf.com`.

   In this description, you must provide a "user manual" describe how to launch the project, describe the command line options, error messages, ... and show the result on some images.

The project can be done by groups of 2 students.

Groups must not copy on each other.
All groups must work on their own!!!