UFAZ DSA Final Exam - C Project

Manipulating Bitmap (BMP) images

You can download an example BMP image there: `https://tinyurl.com/ya53jpv6`. 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 0C 48, which corresponds to 1 228 872 in decimal, *i.e.* the size of the file (must be modified this in the project).
- 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.

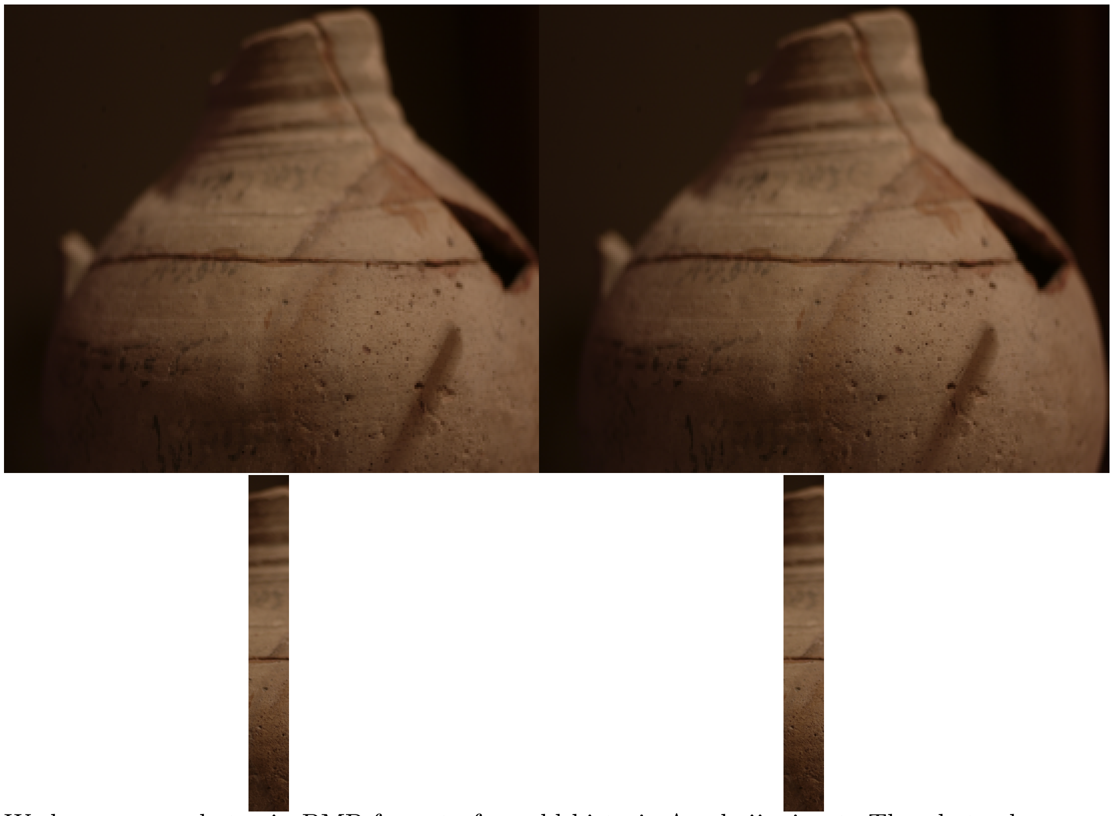
Now should come the colour table but... because we are using 32 bits per pixels, the colour table is not needed :-)

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 Context - flatten a cylinder by strips extraction



We have many photos in BMP format of an old historic Azerbaijani pot. The photos have been taken by rotating the pot of a couple of degrees between each photo. Scientists would like to study this piece of art and extract data from the inscriptions on this pot and for this, it would be very useful to show the pot as if it was flat (and not cylindrical).

So, in order to "flatten" the pot, an easy way is to extract a strip/band from the middle of each photo and join them together afterwards. The scientist must be able to decide the width to be extracted from the middle of the photo

We would like you to write a program **extractStrip** that asks the user how many pixels he would like to extract from the middle of the image. Example: if the user gives 100, this program should extract [-50,+50] starting from the middle of the photo.

To do this, you have to load the file into memory, then find out the width of the image by looking into the BMP header, then create a new file. In this new file, you need to modify the header for the new size of the file and the width of the image (100, for example) and copy the 100 middle pixels of the original BMP image into the new BMP image, and then save the file under the same name followed by `S100` to indicate that it's a strip of 100 pixels.

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.

3 Steps

1. Understand the BMP format of an image.
2. Write the program **extractStrip**.
   (a) ask the user for the width of the strip or (better) get it from the command line if the user types `extractStrip -100 filename.bmp`
   (b) optional: if the file is called with wrong arguments, or if the filename does not correspond to a BMP file (wrong header), write an error message or a "help" text that will also show if you type `extractStrip -h` or `extractStrip --help` (cf. `http://www.catb.org/esr/writings/taoup/html/ch10s05.html` to understand how unix/linux options work).
   (c) Open the file and store its contents into memory, by doing a malloc of the correct size.
   (d) Allocate another chunk of memory to receive the image of the strip (header size + (strip size * height of the image)*4 because there will be 4 bytes / pixel).
   (e) Copy the header of the original image into the new image.
   (f) Modify the values of the header to reflect the new size of the image and its new width.
   (g) Copy the correct pixels corresponding to the desired strip of the original image into the new image.
   (h) Store the new image into a file whose name will be `original_name_Sxx.bmp` where xx is the size of the extracted strip.
3. Write a shell script that uses the program **extractStrip** on all the files of a directory (a simple one can be: `for i in ‘ls‘ ; do ; extractStrip -100 $i ; done`)

I think this program would take me (Pierre Collet) 2 to 3 full hours to write so... start early: it could take you a bit longer to get it to work correctly.

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

The best program will be used for real by Professor Bahlul Ibrahimli to study Azerbaijani archaeological items.