MiniPNG: Writing a Parser for an Image File Format

Olivier Levillain

CSC 4205

The goal of this lab is to write a parser handling MiniPNG images. MiniPNG is a simplified version of the PNG file format (which is defined in RFC 2083).

The lab is composed of three parts:

- the first one consists in parsing the structure of a MiniPNG file;
- the second part is about displaying a black-and-white image interpreted by the parser;
- finally, the third one aims at extending the parser to other kinds of images (color, gray-level, palette).

The "mandatory" part of the lab is composed of the first 5 questions, which are graded on 16 points. If you are motivated, you can work on the lab after the session.

You can use the programming language of your choice, but you must make sure your work includes the information required to compile and run your project.

Obviously, you are expected to think beyond the functional point of view for this project and try to implement the specification in a secure way.

Remarks on your assignment:

- Your are expected to provide a program answering to the different questions that you will enrich with each question[1].
- It is useless to send me a compiled program, since I will not run it. Instead, you must send me the source code and instructions to compile and run it.
- Your program must take an argument on the command line which is the name of the file to handle.
- Your program should halt with an error message (a non null exit code) when you encounter an unexpected situuation.
- For the `A.mp` sample, which belongs to the sample files, your program should produce the following output:

```
Width = 8
Height = 10
Pixel Type = 0

Comments = La lettre A

  XXXX
 X    X
X      X
X      X
X      X
XXXXXXXX
X      X
X      X
X      X
X      X
```

[1] For the 5th question, you are not required to send the scripts to generate the image, but the resulting image is sufficient.

Please send the current version of your work at the end of the lab at `olivier.levillain@telecom-sudparis.eu` with your name and the source code. If you want to send improved version later, you can send them to the same address.

1 Black-and-White Images with MiniPNG

A MiniPNG image starts with a magic number (the 8 bytes representing the 'Mini-PNG' string), followed by "blocks". Each of this block follows the following format:

Offset	Field Name	Field Size	Field Description
0	Block Type	1 byte	A character identifying the block type
1	Block Length	4 bytes	An integer indicating the size l of the block content
5	Block Content	l bytes	The content of the block, which is type-dependent

First, let us describe the blocks required for a MiniPNG Black-and-White image:

- A Header block (type `H`);
- Zero, one or more Comment blocks (type `C`);
- One or more Data blocks (type `D`).

Block `H` Description

The content of the `H` block is the following:

Offset	Field Name	Field Size	Field Description
0	Image Width	4 bytes	An integer describing the width in pixels
4	Image Height	4 bytes	An integer describing the height in pixels
8	Pixel Type	1 byte	0 = black-and-white, 1 = gray levels, 2 = palette, 3 = 24 bits colors

In a first time, we will only consider images with Pixel Type 0 (black-and-white).

Block `C` Description

The content of Comment blocks is simply a character string composed of displayable ASCII chars.

Block `D` Description

The content of data blocks must be concatenated to obtain the image bitmap.

With black-and-white pixels, each pixel is encoded as a bit (0 means black, 1 means white). Other pixel types can be ignored for the moment. Pixels are stored from top to bottom, then left to right.

2 First Steps with MiniPNG

Reminder: the introduction contains indications on how your program is supposed to behave. Please read them carefully!

Question 1. Write a parser reading a MiniPNG file and displaying the image metadata: width, height, pixel type.
Here is the expected output for the `A.mp` file:

```
% ./minipng A.mp
Width: 8
Height: 10
Pixel Type: 0 (back-and-white)
```

Question 2. Enrich your program so it also displays the comments present in the file.
The expected output for `A.mp` is:

```
% ./minipng A.mp
Width: 8
Height: 10
Pixel Type: 0 (black-and-white)
Comments:
"La lettre A"
```

Question 3. Add a check in your program to ensure the data obtained from `D` blocks are consistent with the expected number of bytes (with regards to the metadata from the header).

3 Displaying B&W Images

Question 4. For black-and-white images, add code to your program to display the image in the console, by drawing white pixels with `X` and black pixels with spaces (or the opposite). The expected result for `A.mp` is given in the introduction.

Question 5. Write a valid MiniPNG vile representing the first letter of your name[2].

4 More Pixel Types

Gray-Level and 24-bit True Color Images

In Grey-level images, pixels are encoded on 1 byte each (`0x00` being black and `0xff` white). With 24-bit colors, each pixel is represented by 3 bytes, one for each of the RGB (Red, Green, Blue) components.

Question 6. Adapt your program to support the new modes, with regards to questions 1 and 3.

Question 7. Improve your code so it supports displaying these new image types. Alternatively, if you are not comfortable with a graphics library, you can write a MiniPNG to "portable pixmaps" converter (`https://fr.wikipedia.org/wiki/Portable_pixmap`), so you can convert the images and display them using a standard image reader.

Gestion de la palette

To save space, if your image uses only a limited set of colors, you can use a palette to describe up to 256 colors. To this aim, we use a new block type, `P`, which contains n RGB entries, each one encoded on 3 bytes.

The first entry will be the 0 entry, and each pixel in the bitmap will be represented by a byte, which is the index of the color entry.

Question 8. Add support for the palette in yor parser and print the different palette entries on the console.

Question 9. Implement the display function for palette images.

[2]Obviously, Alices and Arthurs in the classroom must choose another letter or a figure, instead of reusing the `A.mp` sample.