Assignment 2

SDL Video Player

Before you begin...

This assignment builds on the code you wrote for Assignment 1. You will be writing a program that takes the decompressed PPM data and displays it on the screen. You will only require the basic decompression functionality for this; you don't need scaling or tweening to be working.

A partial solution for Assignment 1 will be released on Moodle for the use of those students who did not get decompression working. You are not required to use this solution; if your Assignment 1 submission could successfully decompress RLE files, it will probably be easier if you use your own code.

Task 1. Video player

Implement `ppmplayer`, which plays a sequence of PPM files as a video using the SDL library.

`ppmplayer` has one command-line argument, representing the delay (in milliseconds) between frames. Delays should not be allowed to accumulate - a 50 frame video played with a delay of 20 should take 1 second to play! After the video has finished playing, the program should close.

`ppmplayer` should take its input as a sequence of PPMs on `stdin`, separated by an integer -1. This means that you should be able to use `ppmplayer` in conjunction with your RLE decompressor to play RLEplay files using the following syntax:

```
rledecode myvideo.rle - | ppmplayer 10
```

This line would decode `myvideo.rle` and send its output into `ppmplayer`, to be played at the rate of 10 milliseconds per frame.

Make sure you have this stage working before moving on to the next task. Even if you do not attempt the next section, you can still get 75% on this assignment if the basic display functionality is complete and your code is of high quality.

Task 2. Manipulating the image

Real video players (and televisions) have several controls which can alter how the image looks for easier viewing, or simply for the viewer's preference.

We would like to add the ability to alter the following about the way the video is displayed:

- The overall brightness of the image
- The contrast between light and dark parts of the image
- The *saturation* of the colours displayed. High saturation means very vivid and distinct colours, while low saturation means that the colours are pale – the ultimate in low colour saturation is a black and white image.

Give `ppmplayer` three additional, optional command-line arguments which should be integers between 0 and 100, which represent the values for brightness, contrast, and colour saturation. In each case, 0 represents low, 50 represents "unchanged from the original image" and 100 represents "as high as possible".

That means that:

- `ppmplayer 25 50 50 50` should produce identical output to `ppmplayer 25`.
- `ppmplayer 25 80 50 50` would play the file with the colours adjusted to make the image brighter.
- `ppmplayer 25 50 25 50` would play the file with reduce contrast between light and dark.
- `ppmplayer 25 80 25 70` would play the file with enhanced brightness, reduced contrast, and enhanced colour saturation.

If you only implement one or two of these adjustments, your program should still accept five command line arguments and simply ignore the ones it doesn't use.

The output from this part will be marked subjectively (and leniently) by looking at the video rather than examining the output digitally. This is easier for the markers and also means that you don't have to worry about whether your output matches some predetermined standard. If your manipulations of the colour values seem reasonable, your output with maximum, minimum, and mid-range values looks right, and your manipulations don't distort the image, then the marks will be paid.

Hint: an alternative colour model, such as HSV or HSI, may make doing these transformations easier!

Submission instructions

You must submit a single archive file in gzipped tar format (`.tar.gz`) through Moodle. See `info tar` (or consult the internet) for information how to create a gzipped tar archive.

You must include a working `Makefile` with your submission. It should be possible for the marker to build all the executables by changing to the root directory of your submission and running `make` at the command line. The use of `make` and `Makefile`s was covered in lectures in Week 4.

You should include a `README` file, in text format, that describes (at a minimum):

- your name and student ID number
- how to compile and run the program
- what functionality is and is not supported
- and any known bugs or limitations.

This information is to help your marker build and run your program. That means that it is to your advantage if it is accurate: don't claim that you have implemented something that's not actually there.

Marks breakdown

Marks for this assignment are broken down as follows:

- Basic functionality, including functional correctness, memory management, error handling, robustness, security, usability etc.: **8 marks**
- Extended functionality (brightness, saturation, contrast): **5 marks**
- Code quality (breakdown into functions, use of headers, readability, adequacy of comments, etc.): **5 marks**
- Adherence to submission guidelines; use of `make`: **2 marks**