Lab 5: File I/O and Structs

Due date: Friday 9 February before 6:30pm.

Introduction

The purpose of this lab is to give you some practice using structs and reading binary data from files. To get started, run a `git pull` in your repository to download the lab 5 starter code.

We've provided the Makefile for this lab; you shouldn't need to change it. Simply run `make` to compile the program.

Introduction: Bitmap files

A *24-bit bitmap file* is a simple image file format that consists of two main parts:

1. The file metadata.
2. The *pixel array*, storing numbers corresponding to the blue, green, and red colour values of each pixel in the image (as a number between 0 and 255).

(Note: the actual file format consists of an optional third section after the pixel array, but we'll ignore that section for the purpose of this lab.)

At fixed locations in the file metadata, there are three important integers (each stored with exactly 4 bytes):

- At byte offset 10-13, the offset in the bitmap file where the pixel array starts.
- At byte offset 18-21, the width of the image, in pixels.
- At byte offset 22-25, the height of the image, in pixels.

Suppose our bitmap image has height m and width n; we'll always assume in this lab that the width n is a multiple of 4, which simplifies the byte layout in the file a little. For this image, the pixel array stores exactly 3nm bytes, in the following way:

- Each group of 3 bytes represents a single pixel, where the bytes store the blue, green, and red colour values of the pixel, in that order.
- Pixels are grouped by row. For example, the first 3n bytes in the pixel array represent the pixels in the top-most row of the image.

That's all you need for this lab, but if you're curious about learning more about the bitmap file format, you can start here.

Task 1: Reading bitmap metadata

In the starter code, we have provided a `main` function in `bitmap_printer.c` that opens a bitmap file for processing, and a separate file `bitmap.c` that implements the required functions for working with bitmaps.

Your first task is to use the description of the bitmap metadata to implement `read_bitmap_metadata`. Don't just try to read in the entire bitmap file! Use `fseek` to navigate to different byte locations so that you only read in the three required integers.

We've provided in the starter code two sample bitmap files you can use to test your work. It's quite possible for you to go out and find your own bitmap images, but beware that there are many variations of this file format, and

we're only covering the **24-bit** version in this lab, so you might need to convert new images to this format yourself (and crop to ensure the width is a multiple of 4).

Task 2: Reading in pixels

Now, you'll read in all of the pixels into memory, using a common pattern for storing a two-dimensional array on the heap. Read through the comments in the starter code carefully to complete this part.

Note that because each pixel colour value (blue, green, or red) is stored in just one byte, the type we use is `unsigned char`, which is guaranteed to represent the numbers 0-255.

Task 3: Cleaning up

Don't forget about tidying up the program's resources! Add some code to the bottom of the `main` function in `bitmap_printer.c` to close the bitmap file and free all of the program's dynamically-allocated memory.