98-008 Homework 1: PPM Parsing

Cooper Pierce `cppierce@andrew.cmu.edu`
Jack Duvall `jrduvall@andrew.cmu.edu`

Fall 2022

Overview

The goal of assignment is to get you used to basic Rust constructs and syntax, filling out the core of a program that actually does something useful! We hope this will give you familiarity with Rust's development workflow and make you appreciate the language.

This assignment will have you write a basic parser for a simple image format known as PPM.

PPM Images

PPM is a very simple image format, consisting of just:

- A "magic number" to distinguish it from other files
- Width and height of the image
- The maximum intensity value of
- Pixels as packed RGB pixel values, read sequentially row-by-row from the top left of the image.

`http://ailab.eecs.wsu.edu/wise/P1/PPM.html` outlines the format; for simplicity, we'll summarize it again here. Note that we'll be using the binary format, where each pixel channel takes up exactly one byte.

PPM Format Specification

`PPM`	::=	`MagicHeader`
		`CommentsAndWhitespace` `Width`
		`CommentsAndWhitespace` `Height`
		`CommentsAndWhitespace` `Maxval`
		"$\n$" `Pixels`
`MagicHeader`	::=	"$P6\n$"
`CommentsAndWhitespace`	::=	("$\t$" | "$\n$" | "$\x0C$" | "$\r$" | " " | ("#" `Comment` "$\n$"))
		`CommentsAndWhitespace`
`Comment`	::=	((Any character except for "$\n$") `Comment`) | ϵ
`Width`	::=	(("0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9") `Width`) | ϵ
`Height`	::=	`Width`
`Maxval`	::=	`Maxval`
`Pixels`	::=	Exactly $w * h * 3$ bytes, where w and h are the parsed width and height

How to read this table: groups being right next to each other means the text they match should be concatenated. The | operator means that either the left or right character can be accepted.

Characters in “” means only that exact string is accepted. ϵ matches the empty string. Height and Maxval being the same as width does not mean the values should be the same; only that the patterns are matched the same.

PPM Image Example

An example would probably help understand the above specification a bit better:

```
P6
# I'm a PPM file! You can tell by the magic header
3 3 # This first number is the width, and the second is the height
# This next number is the maximum intensity of each pixel
# Exporting as "raw" in GIMP always gives 255, which makes sense; this is the
# maximum value of a u8. After this maxval, we are only allowed a single newline
# before the pixels start. These pixels can be arbitrary bytes!
255
<<<<<<<<<aaaaaaaaa~~~~~~~~~
```

Copy-pasting this text into a file ending with `.ppm` and opening it in a compatible image-viewing program should give a 3x3 image with 3 horizontal gray stripes.

What You Will Write

You will be writing code that parses the metadata in the PPM header, up until the pixels.

You have been provided enum variants representing a state machine that interprets the PPM format specification as found in the previous section, as well as code to read the pixels into an appropriately-sized buffer and display them on screen.

The code you will write will effectively be a big `match` expression that drives the state machine transitions. You are free to structure your code however you wish, including editing outside the desired area, so long as the testcases in the main file pass.

The code you should read/write will all be in the `src/ppm.rs` file. You may optionally read `src/main.rs` to see how your code is used, if you wish. Please do not change any other files, including `Cargo.toml` or any of the `*.ppm` files used for testcases.

You will submit to gradescope a tarball with the same directory layout as the handout tarball, just with your code filled in.

As always, if you have any questions, ask in Discord, email the course staff, or go OH. Have fun with the assigment!