98-008 Homework 3: `egrep`

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

Fall 2022

Overview

The goal of assignment is to apply what we've talked thus far in class, and start to apply it to a larger problem, with some more involved existing code. It also involves IO, which you'll likely use in your project, as well as applying some of the topics, like traits, that we've discussed. A good understanding of the standard library will also help! Documentation for the standard library can be found at `https://doc.rust-lang.org/std/` and is an invaluable resource.

`egrep`

In this assignment you'll be implementing components of the utility `egrep` (this is the same as `grep -E`, but aliased as one command—the regex format is a bit more sane than by default). Most of your work will be to utilise the existing NFA based regex matching engine to actually interact with command line input, and producing the corresponding output.

Basic Matching

To start with, you'll need to implement the logic in `main` and some parts of `Matcher` to be able to find matches in the first place. No need to worry about parsing patterns or writing an efficient matching engine—we've done that for you! We suggest you write a function like

```
fn get_files<'a>(
    files: impl Iterator<Item = &'a str>,
) -> Vec<io::Result<(&'a str, Box<dyn BufRead>)>> {
    todo!()
}
```

In order to handle the input files, because we need to handle standard in.

If we're ever given a file name of `-`, or no files are provided, we want to read input from standard in instead. Other inputs after the pattern should be treated as relative paths, and you can read from these directly. In the case of non-existent files or other IO related errors, you can print any reasonable diagnostic, but you should exit with a non-zero exit code.

You will probably find `File` and `stdin` useful. Likewise, the `Read` and `Write` traits from the standard library (or their buffered versions) are likely to be helpful reading.

Your program should essentially behave the same as your system's install (sorry, Windows users; perhaps consult andrew) of `egrep`. Specifically:

- Each matching line, and only these lines, should be written to standard output
- If more than one file argument appears, prefix each such line with the filename followed by a colon

So if we ran `egrep foo A B` and both A and B contained one line each consisting of foo, then we'd expect to see

```
A:foo
B:foo
```

Note that there's no space after the colon! You might not care but Gradescope does.

You can assign standard input any reasonable name for the purpose of printing its filename.

Highlighting Matches

As an extension, now highlight the specific matching region! This involves a little bit more implementation in the `Matcher`, but as before, most of the work has already been done.

How you handle the printing here is up to you; the only requirement is that the match is bolded and a different colour. We suggest the https://docs.rs/termion/latest/termion/ crate, but you can use any crate you wish, or even handle the ANSI escape codes by hand (a bit ugly, but it works). Note that Gradescope is a Linux environment, so whatever you do has to work for this, so if you're on Windows don't use something Windows only.

What you print should roughly correspond to the results for `egrep --color`, but as this is less standard, it doesn't need to be exact.

Submitting

Submission will be on Gradescope.

You should submit a zip file containing the whole crate rooted at the directory where `Cargo.toml` appears (e.g., run `zip submission.zip Cargo.toml src/*`) and upload that.