Description

The purpose of this project will be to build experience in Rust programming while exploring algorithms to process regular expressions.

You will build a Rust program to validate strings against a regular expression (definition below). The program will:

a) Accept a regular expression from the command line.
b) Build an internal representation of the state diagram for the regular expression.
c) Output to `stdout` the Graphiz definition of the state diagram.
d) Read lines from `stdin`. The reason for using `stdin` is that you can either type in lines to test with or produce a text file that you redirect into the program.
e) Each line from the file be a string that will be processed by the state machine.
f) If the string is accepted by the state machine (it matches the regular expression), print "Accept" and the string to `stderr`.
g) If the string is rejected by the state machine (it doesn't match the regular expression), print "Reject" and the string to `stderr`

There are a variety of different ways to solve this problem. In particular I will be looking for good efficient use of Rust collections and data structures. Further, you might want to think about parsing the regular expression (RegEx) using a context-free grammar. It's not required but might make your life simpler (or not).

The Regular Expression

The following characters and sequences will need to be supported in your regular expression. Note, I suggest starting with the basic characters, get those working before adding support for additional characters.

- $\Sigma = \{a..z, 0..9, blank\}$ – The symbols of the language you will accept

- Regular expression operators

    - The concatenation operation is indicated by symbols (or parenthetical or bracketed expressions) placed next to each other.

      For example, the RegEx '`ab`' would accept any string containing the symbol `a` followed by a `b` symbol.

    - '*' – the star operator as defined on regular expressions (zero or more occurrences of the preceding character).

      For example, the RegEx '`a*b`' would accept a string containing zero or more `a` symbols ending in a `b` symbol.

    - '|' – bar character indication the union operator as defined on regular expressions

      For example, the RegEx '`aa|qq|5`' would accept a string containing either `aa`, `qq` or the `5` symbol.

- ‘()’ – a parenthetical expression indicating that the contents should be considered a single unit

  For example, the RegEx ‘(aa|qq|5)*’ would accept any number of repetitions of the string from the preceding example.

- Additional, extra credit operators
  - ‘+’ – the plus character indicating *one* or more occurrences of the preceding symbol or expression
  - ‘\w’ – indicating any alphabetic character (a-z)
  - ‘\d’ – indicating any numeric character (0-9)

Due Date and Deliverables

The project is due by 5:30PM, Thursday, November 5. Late work will not be accepted.

I will be expecting, at a minimum, a source file, associated Rust files (e.g. Cargo.toml), and a `README` file describing how to build the project. Details of the operation of the program including any limitations should be built included in the source code as comments. I will be generating Rustdoc for the program and this information will need to appear in the generated documentation. My preference would be a link to a GitHub (Bitlocker, GitLab, etc.) repository that I can clone to see the code (or include it in your class repository), but I will accept the aforementioned files submitted into the Oaks dropbox.

Rubric

The project will be worth 100 points with a possible 15-point bonus allocated as follows:

1. 60 Points – Operation

   - (5 Pts) Does the program build correctly?
   - (5 Pts) Does the program run and produce useful, readable, output?
   - (10 Pts) Does the program accept the symbols of the language specified?
   - (10 Pts) Does the program handle command line and stdin input as specified?
   - (30 Pts) Does the program support each of the RegEx operators listed?

2. 40 Points – Programming Style

   - (10 Pts) Is the code well commented and readable?
   - (10 Pts) Are all needed files/libraries provided?
   - (10 Pts) Is sufficient documentation generated by Rustdoc to allow an understanding off the use of the program?
   - (10 Pts) Does the program represent the state machine for the RegEx in an efficient form demonstrating the use of the Rust library.

3. 15 Pts – Bonus for supporting additional RegEx operators as discussed above.