CptS 360 (System Programming)
Unit 17: Regular Expressions

Bob Lewis

School of Engineering and Applied Sciences
Washington State University

Spring, 2022

Motivation

- Regular expressions ("RE"s) are extremely useful in system (and other) programming.
- Facility with REs is the mark of an experienced system programmer.
- Regular expressions are part of the POSIX standard.

References

- https://remram44.github.io/regex-cheatsheet/regex.html
  (This includes a comparison of several RE "flavors".)
- *regex(3)*
- Friedl, Jeffrey E. F., *Mastering Regular Expressions*, O'Reilly, Inc., 3rd. ed. (2006)

Regular Expression Flavors

- Perl/PCRE ("Perl Compatible Regular Expressions")
  See *pcre(3)* (which differs slightly from Perl itself). Used by Perl (of course), *grep(1)* (with the "`-P`" option), and several text editors and IDEs. It's powerful, but may be overkill.
- Python's `re` package
  See "`$ pydoc3 re`". Used by the Python language. ("Perl inspired")
- POSIX (Basic)
  See *regex(3)*. Used by *grep(1)* and *sed(1)*.
- POSIX (Extended)
  Also see *regex(3)*. Used by *awk(1)*, *less(1)*, *egrep(1)*, and *sed(1)* (with the "`-E`" option).
- Vim
  Used by the *vim(1)* editors.

One Curmudgeon's Opinion

"I define UNIX as 30 definitions of regular expressions living under one roof."
– Donald Knuth (Computer Scientist Supreme)

The number is a little high, but he has a point. We'll cover the two POSIX flavors, but the others are upwardly-compatible with these features.

What is an RE?

- An RE is a string (a *pattern*) of one or more *components* specifying a sequence of characters to be matched in a *target* string.
- The simplest components:
  - a letter a-z or A-Z
  - a decimal digit 0-9
  - any other character not used as a metacharacter (below)
  - any metacharacter "escaped" with a preceding '\'
- For a RE match to succeed, each component in the RE must either match one or more characters in the target or indicate a boundary, non-boundary, repetition, alternation (choice from one of several components), or group.

(see `demos/dn_regexplore`)

Character Class Components

These match one of several characters:

component	matches any ...
.	character except newline
[*seq*]	character in *seq* (alternation)
[^*seq*]	character not in *seq*
\w	identifier (alphanumeric and "_") character
\W	non-identifier character
[*ch0*-*ch1*]	character between *ch0* and *ch1*, inclusive
\s	whitespace (e.g., ' ' or '\t') character
[[:*keyword*:]]	character belonging to the *keyword* class (next slide)
[^ [:*keyword*:]]	character not belonging to the *keyword* class

Keyword Character Classes

keyword	component matches any...
`upper`	upper case letter
`lower`	lower case letter
`space`	whitespace character (same as '\s')
`alpha`	alphabetical character
`alnum`	alphanumeric character
`digit`	decimal digit
`xdigit`	hexadecimal digit
`punct`	punctuation mark

You can combine keyword and non-keyword character classes, so "`[[:digit:][:punct:]X]`" will match a decimal digit, a punctuation mark, or the letter '`X`'.

Boundary Components

Instead of characters, these match boundaries before, after, and between characters.

component	boundary
^	start of the line
$	end of the line
\<	start of word
\>	end of word
\b	start or end of word
\B	non-start or non-end of word

These are all "zero-width", so use them with non-boundary components or *regex(3)* will get confused.

Repetition

These symbols indicate a repetition of the previous component:

symbol	indicates this many repetitions
\?	0 or 1 (Basic)
?	0 or 1 (Extended)
*	0 or more
\+	1 or more (Basic)
+	1 or more (Extended)
\{n\}	exactly n (Basic)
{n}	exactly n (Extended)
\{n,m\}	between n and m, inclusive (Basic)
{n,m}	between n and m, inclusive (Extended)
\{n,\}	n or more (Basic)
{n,}	n or more (Extended)

Alternation and Grouping

These operators allow you to combine alternative REs and define and match previous RE groups.

operator	**means**
re0|*re1*	match either *re0* or *re1* (Basic)
re0|*re1*	match either *re0* or *re1* (Extended)
\(*re*\)	define a group matching *re* (Basic)
(*re*)	define a group matching *re* (Extended)
n	match previously-defined group *n*

Within a match, group 0 is always the whole match, group 1 is the leftmost group within the match, etc.

Some RE Applications

Here are a couple of RE applications:

- *scramble*
  demonstrates an (alleged) research result that says that people can understand text if the words are scrambled as long as the first and last letters are preserved.
  (run `demos/dn_scramble`)
  Note the use of RE matches for substitution.
- *willpower*
  uses REs to analyze a text of a play (courtesy of Project Gutenburg) for stage directions.
  (run `demos/dn_willpower`)

and, for fun and to test your RE expertise, take a look at `re_crossword.pdf` in the lecture notes directory.