COMP284 Scripting Languages

Lecture 4: Perl (Part 3)
Handouts (8 on 1)

Ullrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

Contents

Regular expressions: Motivation

Suppose you are testing the performance of a new sorting algorithm by measuring its runtime on randomly generated arrays of numbers of a given length:

```
Generating an unsorted array with 10000 elements took 1.250 seconds
Sorting took 7.220 seconds
Generating an unsorted array with 10000 elements took 1.243 seconds
Sorting took 10.486 seconds
Generating an unsorted array with 10000 elements took 1.216 seconds
Sorting took 8.951 seconds
```

Your task is to write a program that determines the average runtime of the sorting algorithm:

```
Average runtime for 10000 elements is 8.886 seconds
```

Solution: The regular expression `/^Sorting took (\d+\.\d+) seconds/` allows us to get the required information

⇝ Regular expressions are useful for information extraction

Regular expressions: Motivation

Suppose you have recently taken over responsibility for a company's website. You note that their HTML files contain a large number of URLs containing superfluous occurrences of '`..`', e.g.

```
http://www.myorg.co.uk/info/refund/../vat.html
```

Your task is to write a program that replaces URLs like these with equivalent ones without occurrences of '`..`':

```
http://www.myorg.co.uk/info/vat.html
```

while making sure that relative URLs like

```
../video/disk.html
```

are preserved

Solution: `s!/[^\/]+/\.\.!!;` removes a superfluous dot-segment

⇝ Substitution of regular expressions is useful for text manipulation

Regular expressions: Introductory example

```
\Ahttps?:\/\/[^\/]+\/.\w.\/(cat|dog)\/\1
```

- `\A` is an assertion or anchor
- `h`, `t`, `p`, `s`, `:`, `\/`, `c`, `a`, `t`, `d`, `o`, `g` are characters
- `?` and `+` are quantifiers
- `[^\/]` is a character class
- `.` is a metacharacter and `\w` is a special escape
- `(cat|dog)` is alternation within a capture group
- `\1` is a backreference to a capture group

Pattern match operation

- To match a regular expression *regexpr* against the special variable `$_` simply use one of the expressions `/`*regexpr*`/` or `m/`*regexpr*`/`
  - This is called a pattern match
  - `$_` is the target string of the pattern match
- In a scalar context a pattern match returns true (1) or false ('') depending on whether *regexpr* matches the target string

```
if (/\Ahttps?:\/\/[^\/]+\/.\w.\/(cat|dog)\/\1/) {
  ... }

if (m/\Ahttps?:\/\/[^\/]+\/.\w.\/(cat|dog)\/\1/) {
  ... }
```

Regular expressions: Characters

The sequence of

- a
- n a backslash:

that matches exactly this sequence of characters occurring as a substring in the target string

```
$_ = "ababcbcdcde";
if (/cbc/) { print "Match\n"} else { print "No␣match\n" }
```

Output:

```
Match
```

```
$_ = "ababcbcdcde";
if (/dbd/) { print "Match\n"} else { print "No␣match\n" }
```

Output:

```
No match
```

Regular expressions: Special variables

- Often we do not just want to know whether a regular expession matches a target string, but retrieve additional information
- The special variable `$-[0]` can be used to retrieve the start position of the match

  Note that positions in strings are counted starting with 0
- The special variable `$+[0]` can be used to retrieve the first position after the match
- The special variable `$&` returns the match itself

```
$_ = "ababcbcdcde";
if (/cbc/) { print "Match␣found␣at␣position␣$-[0]:␣$&\n"}
```

Output:

```
Match found at position 4: cbc
```

Regular expressions: Special escapes

There are various special escapes and metacharacters that match more then one character:

.	Matches any character except \n
\w	Matches a 'word' character (alphanumeric plus '_', plus other connector punctuation characters plus Unicode characters
\W	Matches a non-'word' character
\s	Match a whitespace character
\S	Match a non-whitespace character
\d	Match a decimal digit character
\D	Match a non-digit character
\p{*UnicodeProperty*}	Match *UnicodeProperty* characters
\P{*UnicodeProperty*}	Match non-*UnicodeProperty* characters

Regular expressions: Unicode properties

- Each unicode character has one or more properties, for example, which script it belongs it
- \p{*UnicodeProperty*} matches all characters that have a particular property
- \P{*UnicodeProperty*} matches those that do not
- Examples of unicode properties are

Arabic	Arabic characters
ASCII	ASCII characters
Currency_Symbol	Currency symbols
Digit	Digits in all scripts
Greek	Greek characters
Han	Chinese kanxi or Japanese kanji characters
Space	Whitespace characters

See http://perldoc.perl.org/pe

Regular expressions: Character class

- A character class, a list of characters, special escapes, metacharacters and unicode properties enclosed in square brackets, matches any single character from within the class, for example, [ad\t\n\-\\09]
- One may specify a range of characters with a hyphen -, for example, [b-u]
- A caret ^ at the start of a character class negates/complements it, that is, it matches any single character that is not from within the class, for example, [^01a-z]

```
$_ = "ababcbcdcde";
if (/[bc][b-e][^bcd]/) {
   print "Match␣at␣positions␣$-[0]␣to␣",$+[0]-1,":␣$&\n"};
```

Output:

```
Match at positions 8 to 10: cde
```

Quantifiers

- The constructs for regular expressions that we have so far are not sufficient to match, for example, natural numbers of arbitrary size
- Also, writing a regular expressions for, say, a nine digit number would be tedious

This is made possible with the use of quantifiers

*regexpr**	Match *regexpr* 0 or more times
regexpr+	Match *regexpr* 1 or more times
regexpr?	Match *regexpr* 1 or 0 times
regexpr{*n*}	Match *regexpr* exactly n times
regexpr{*n*,}	Match *regexpr* at least n times
regexpr{*n*,*m*}	Match *regexpr* at least n but not more than m times

Quantifiers are greedy by default and match the longest leftmost sequence of characters possible

Quantifiers

*regexpr**	Match *regexpr* 0 or more times
regexpr+	Match *regexpr* 1 or more times
regexpr?	Match *regexpr* 1 or 0 times
regexpr{*n*}	Match *regexpr* exactly n times
regexpr{*n*,}	Match *regexpr* at least n times
regexpr{*n*,*m*}	Match *regexpr* at least n but not more than m times

Example:

```
$_ = "Sorting␣took␣10.486␣seconds";
if (/\d+\.\d+/) {
   print "Match␣at␣positions␣$-[0]␣to␣",$+[0]-1,":␣$&\n"};
$_ = "E00481370";
if (/[A-Z]0{2}(\d+)/) {
   print "Match␣at␣positions␣$-[1]␣to␣",$+[1]-1,":␣$1\n"};
```

Output:

```
Match at positions 13 to 18: 10.486
Match at positions 3 to 8: 481370
```

Quantifiers

Example:

```
$_ = "E00481370";
if (/\d+/) {
   print "Match␣at␣positions␣$-[0]␣to␣",$+[0]-1,":␣$&\n"};
```

Output:

```
Match at positions 1 to 8: 00481370
```

- The regular expression \d+ matches 1 or more digits
- As the example illustrates, the regular expression \d+
  - matches as early as possible
  - matches as many digits as possible

  ⇝ quantifiers are greedy by default

Revision

Re[illegible]

- [illegible] pressions
- Chapter 8: Matching with Regular Expressions

of

R. L. Schwartz, brian d foy, T. Phoenix:
Learning Perl.
O'Reilly, 2011.

- http://perldoc.perl.org/perlre.html
- http://perldoc.perl.org/perlretut.html
- http://www.perlfect.com/articles/regextutor.shtml