COMP284 Scripting Languages

Lecture 5: Perl (Part 4)

Handouts

Ullrich Hustadt

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

Contents

Regular expressions: Capture groups and backreferences

- We often encounter situations where we want to identify the repetition of the same or similar text, for example, in HTML markup:

```
<strong> ... </strong>
<li> ... </li>
```

- We might also not just be interested in the repeating text itself, but the text between or outside the repetition

- We can characterise each individual example above using regular expressions:

```
<strong>.*<\/strong>
<li>.*<\/li>
```

but we cannot characterise both without losing fidelity, for example:

```
<\w+>.*<\/\w+>
```

does not capture the 'pairing' of HTML tags

Regular expressions: Capture groups

The solution are capture groups and backreferences

(*regexpr*)	creates a capture group
(?<*name*>*regexpr*)	creates a named capture group
(?:*regexpr*)	creates a non-capturing group
\N, \g*N*, \g{*N*}	backreference to capture group *N* (where *N* is a natural number)
\g{*name*}	backreference to a named capture group

Examples:

```
/Sorting took (\d+\.\d+) seconds/
/<(\w+)>.*<\/\1>/
/([A-Z])0{2}(\d+)/
/(?<c1>\w)(?<c2>\w)\g{c2}\g{c1}/
/((?<c1>\w)(?<c2>\w)\g{c2}\g{c1})/
```

Regular expressions: Capture groups

Via capture variables the strings matched by a capture group are also available outside the pattern in which they are contained

$N	string matched by capture group *N* (where *N* is a natural number)
$+{*name*}	string matched by a named capture group

The matched strings are available until the end of the enclosing code block or until the next successful match

Example:

```
$_ = "Yabba␣dabba␣doo"
if (/((?<c1>\w)(?<c2>\w)\g{c2}\g{c1})/) {
  print "Match␣found:␣$1\n" }
```

Output:

```
Match found: abba
```

Regular expressions: Alternations

- The regular expression *regexpr1*|*regexpr2* matches if either *regexpr1* or *regexpr2* matches
  This type of regular expression is called an alternation
- Within a larger regular expression we need to enclose alternations in a capture group or non-capturing group:
  (*regexpr1*|*regexpr2*) or (?:*regexpr1*|*regexpr2*)

Examples:

```
/Mr|Ms|Mrs|Dr/
/cat|dog|bird/
/(?:Bill|Hillary) Clinton/
```

Regular expressions: Alternations

- The order of expressions in an alternation only matters if one expression matches a sub-expression of another

Example:

```
$_ = "cats␣and␣dogs";
if (/(cat|dog|bird)/) { print "Match␣1:␣$1\n" }
if (/(dog|cat|bird)/) { print "Match␣2:␣$1\n" }
if (/(dog|dogs)/) { print "Match␣3:␣$1\n" }
if (/(dogs|dog)/) { print "Match␣4:␣$1\n" }
```

Output:

```
Match 1: cat
Match 2: cat
Match 3: dog
Match 4: dogs
```

Regular expressions: Anchors

Anchors allow us to fix where a match has to start or end

`\A`	Match only at string start
`^`	Match only at string start (default) Match only at a line start (in `//m`)
`\Z`	Match only at string end modulo a preceding `\n`
`\z`	Match only at string end
`$`	Match only at string end modulo a preceding `\n` Match only at a line end (in `//m`)
`\b`	Match word boundary (between `\w` and `\W`)
`\B`	Match except at word boundary

Example:

```
$_ = "The␣girl␣who\nplayed␣with␣fire\n";
if (/fire\z/) { print "‘fire’␣at␣string␣end\n" }
if (/fire\Z/) { print "‘fire’␣at␣string␣end␣modulo␣\\n\n" }
```

```
‘fire’␣at␣string␣end␣modulo␣\n
```

Regular expressions: Modifiers

Modifiers change the interpretation of certain characters in a regular expression or the way in which Perl finds a match for a regular expression

`/ /`	Default '.' matches any character except '\n' '^' matches only at string start '$' matches only at string end modulo preceding \n
`/ /s`	Treat string as a single long line '.' matches any character including '\n' '^' matches only at string start '$' matches only at string end modulo preceding \n
`/ /m`	Treat string as a set of multiple lines '.' matches any character except '\n' '^' matches at a line start '$' matches at a line end

Regular expressions: Modifiers

Modifiers change the interpretation of certain characters in a regular expression or the way in which Perl finds a match for a regular expression

`/ /sm`	Treat string as a single long line, but detect multiple lines ‘`.`’ matches any character including ‘`\n`’ ‘`^`’ matches at a line start ‘`$`’ matches at a line end
`/ /i`	perform a case-insensitive match

Example:

```
$_ = "bill\nClinton";
if (/(Bill|Hillary).Clinton)/smi) { print "Match:␣$1\n" }
```

Output:

```
Match: bill
Clinton
```

Regular expressions: Modifiers (/ /g and / /c)

Often we want to process all matches for a regular expression, but the following code has not the desired effect

```
$_ = "1␣22␣33";
while (/\d+/) { print "Match␣starts␣at␣$-[0]:␣$&\n" }
```

The code above does not terminate and endlessly prints out the same text:

```
Match starts at 0: 1
```

To obtain the desired behaviour of the while-loop we have to use the / /g modifier:

/ /g	In scalar context, successive invocations against a string will move from match to match, keeping track of the position in the string
	In list context, returns a list of matched capture groups, or if there are no capture groups, a list of matches to the whole regular expression

Regular expressions: Modifiers (/ /g and / /c)

With the / /g modifier our code works as desired:

```
$_ = "11␣22␣33";
while (/\d+/g) { print "Match␣starts␣at␣$-[0]:␣$&\n" }
```

Output:

```
Match starts at 0: 11
Match starts at 3: 22
Match starts at 6: 33
```

An example in a list context is the following:

```
$_ = "ab␣11␣cd␣22␣ef␣33";
@numbers = (/\d+/g);
print "Numbers:␣",join("␣|␣",@numbers),"\n";
```

Output:

```
Numbers: 11 | 22 | 33
```

Read / /g as: Start to look for a match from the position where the last match using / /g ended

Regular expressions: Modifiers (/ /g and / /c)

The current position in a string for a regular expression *regexpr* is associated with the string, not *regexpr*

- ⇝ different regular expressions for the same strings will move forward the same position when used with / /g
- ⇝ different strings have different positions and their respective positions move forward independently

Example:

```
$_ = "ab␣11␣cd␣22␣ef␣33";
if (/\d+/g)     { print "Match␣starts␣at␣$-[0]:␣$&\n" }
if (/[a-z]+/g)  { print "Match␣starts␣at␣$-[0]:␣$&\n" }
if (/\d+/g)     { print "Match␣starts␣at␣$-[0]:␣$&\n" }
```

Output:

```
Match starts at 3: 11
Match starts at 6: cd
Match starts at 9: 22
```

Regular expressions: Modifiers (/ /g and / /c)

A failed match or changing the target string resets the position

```
$_ = "ab␣11␣cd␣22␣ef␣33";
if (/\d+/g) { print "2:␣Match␣starts␣at␣$-[0]:␣$&\n" }
if (/ab/g)  { print "3:␣Match␣starts␣at␣$-[0]:␣$&\n" }
if (/\d+/g) { print "4:␣Match␣starts␣at␣$-[0]:␣$&\n" }
```

Output:

```
2: Match starts at 3: 11
4: Match starts at 3: 11
```

To prevent the reset, an additional modifier / /c can be used

```
$_ = "ab␣11␣cd␣22␣ef␣33";
if (/\d+/g) { print "2:␣Match␣starts␣at␣$-[0]:␣$&\n" }
if (/ab/gc) { print "3:␣Match␣starts␣at␣$-[0]:␣$&\n" }
if (/\d+/g) { print "4:␣Match␣starts␣at␣$-[0]:␣$&\n" }
```

Output:

```
2: Match starts at 3: 11
4: Match starts at 9: 22
```

Generating regular expressions on-the-fly

The Perl parser will expand occurrences of $*variable* and @*variable* in regular expressions

⇝ regular expressions can be constructed at runtime

Example:

```
$_ = "Bart␣teases␣Lisa";
@keywords = ("bart","lisa","marge",'L\w+',"t\w+");
while ($keyword = shift(@keywords)) {
  print "Match␣found␣for␣$keyword:␣$&\n" if /$keyword/i;
}
```

Output:

```
Match found for bart: Bart
Match found for lisa: Lisa
Match found for L\w+: Lisa
Match found for t\w+: teases
```

Binding operator

Perl offers two binding operators for regular expressions

string =~ /*regexpr*/	true iff *regexpr* matches *string*
string !~ /*regexpr*/	true iff *regexpr* does not match *string*

- Note that these are similar to comparison operators not assignments
- Most of the time we are not just interested whether these expressions return true or false, but in the side effect they have on the special variables $*N* that store the strings matched by capture groups

Example:

```
$name = "Dr␣Ullrich␣Hustadt";
if ($name =~ /(Mr|Ms|Mrs|Dr)?\s*(\w+)/) {print "Hello␣$2\n"}
$name = "Dave␣Shield";
if ($name =~ /(Mr|Ms|Mrs|Dr)?\s*(\w+)/) {print "Hello␣$2\n"}
```

```
Hello Ullrich
Hello Dave
```

Pattern matching in a list context

- When a pattern match /*regexpr*/ is used in a list context, then the return value is
  - a list of the strings matched by the capture groups in *regexpr* if the match succeeds and *regexpr* contains capture groups, or
  - (a list containing) the value 1 if the match succeeds and *regexpr* contains no capture groups, or
  - an empty list if the match fails

```
$name = "Dr␣Ullrich␣Hustadt";
($t,$f,$l) = ($name =~ /(Mr|Ms|Mrs|Dr)?\s*(\w+)\s+(\w+)/);
print "Name:␣$t,␣$f,␣$l\n";
$name = "Dave␣Shield";
($t,$f,$l) = ($name =~ /(Mr|Ms|Mrs|Dr)?\s*(\w+)\s+(\w+)/);
print "Name:␣$t,␣$f,␣$l\n";
```

Output:

```
Name: Dr, Ullrich, Hustadt
Name: , Dave, Shield
```

Pattern matching in a list context

- When a pattern match /*regexpr*/g is used in a list context, then the return value is
  - a list of the strings matched by the capture groups in *regexpr* each time regex matches provided that *regexpr* contains capture groups, or
  - a list containing the string matched by *regexpr* each time *regexpr* matches provided that *regexpr* contains no capture groups, or
  - an empty list if the match fails

```
$string = "firefox:␣10.3␣seconds;␣chrome:␣9.5␣seconds";
%performance = ($string =~ /(\w+)\:\s+(\d+\.\d+)/g);
foreach $system (keys %performance) {
   print "$system␣->␣$performance{$system}\n" }
```

Output:

```
firefox -> 10.3
chrome -> 9.5
```

Revision

Read

- Chapter 7: In the World of Regular Expressions
- 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`