Regular expressions

Introduction

- Regular expressions are used in Perl in a number of ways:
  1. Search for a string that matches a pattern, and optionally replacing the pattern found with some other strings
  2. Counting the number of occurrences of a pattern in a string
  3. Split a formatted string (e.g. a date like 02/06/2001) into respective components (i.e. into day, month and year)

Building a Pattern (1/2)

- The **m//** operator
  - Use for pattern matching
- For example, we just need to know if the characters "able" appear in any given string, the pattern is as simple as:

m/able/

Building a Pattern (2/2)

- Using the binding operator =~

```
if ("Capable" =~ m/able/)
{
        print "match!\n";
}
else
{
        print "no match!\n";
}
```

The m// and the Binding Operator (1/4)

- In between the forward slashes // the pattern to match is placed
- If any options are present they are placed at the end after the last slash
- If an expression is explicitly bound to the operator using the =~ or !~ binding operators, that expression is searched for the pattern specified

The m// and the Binding Operator (2/4)

- The binding operator =~ returns a true value if the expression matches the pattern, an empty string (and hence a false value) if otherwise
- !~ simply inverts the logic so that if the expression matches the pattern a false value is returned, a true value otherwise.
- Therefore, the following two expressions are equivalent

```
!($expression =~m/pattern/)
$expression !~m/pattern/
```

The m// and the Binding Operator (3/4)

- The pattern may be interpolated

```
$expression =~m/$var/
```

- If we use //, you may omit the prefix m
- Can also use other symbols in place of // if the pattern is heavily slashed

```
$expression =~m/\/var\/logs\/httpd\/error_log/
```

- This is *leaning toothpick syndrome* (LTS) where a lot of forward and backward slashes are present, making the pattern itself difficult to recognize

The m// and the Binding Operator (4/4)

- We change the symbol to | (say), then the entire pattern suddenly becomes clear:

```
$expression =~m|/var/logs/httpd/error_log|
```

Metacharacters (1/8)

- Metacharacters serve specific purposes in a pattern
- If any of these metacharacters are to be embedded in the pattern literally, we should quote them by prefixing it by \

Metacharacters (2/8)

Regular Expressions

Metacharacter	Default Behaviour
\	Quote next character
^	Match beginning-of-string
.	Match any character except newline
$	Match end-of-string
|	Alternation
()	Grouping and save subpattern
[]	Character class

Metacharacters (3/8)

- For example, **to match a pair of empty parentheses and execute a code block if they can be found**, the code should look like

```
if ($string =~m/\(\)/)
{
  # ...
}
```

Metacharacters (4/8)

- This does not suppress interpolation, however

```
$expression =~m/\Q/var/logs/httpd/error_log\E/
```

- | species alternate patterns where matching of either one of them results in a match
- These patterns are tried from left to right
- The first one that matches is the one taken

Metacharacters (5/8)

- Usually, | are used together with parentheses () to indicate the groupings preferred

```
m/for|if|while/
# A match if either 'for', 'if' or 'while' found

m/a(a|b|c)a/
# A match if either 'aaa', 'aba' or 'aca' found
```

Metacharacters (6/8)

- The parentheses() besides grouping, there is also another usage.
  - If there is a pattern match, the expression matched by a grouped pattern is saved. This is called backtracking
- The . metacharacter matches any character
- By default, it does not match any embedded newline characters in a multi-line string
- However, if the s option of m// is given, embedded newline characters will be matched

Metacharacters (7/8)

```
"a\nb\nc" =~m/a.b/
# Not matched, because . does not match \n

"a\nb\nc" =~m/a.b/s
# Matched with 's' option
```

Metacharacters (8/8)

- The ^ metacharacter matches the beginning of the string, and $ matches the end of the string
- However, if the m option of m// is given, they match the beginning and the end of each line respectively
- This is used to match individual lines inside a multi-lined string.

```
"a\nb\nc" =~m/^a$/
# Not matched

"a\nb\nc" =~m/^a$/m
# Matched
```

Quantifiers (1/4)

- Quantifiers are used to specify how many times a certain pattern can be matched consecutively
- A quantifier can be specified by putting the range expression inside a pair of curly brackets
- The format of which is

```
{m[,[n]]}
```

Quantifiers (2/4)

- Here are the available variations:
  - {m} Match exactly m times
  - {m,} Match m or more times
  - {m, n} Match at least m times but not more than n times

- Eg. Verify if a string is an even number

```
$string = $ARGV[0];
my $retval = ($string =~m/^(\+|-) { 0, 1}
(0|1|2|3|4|5|6|7|8|9){0,}(0|2|4|6|8)$/);
printf("$string is %s an even integer.\n",
$retval?' ':' not ');
```

Quantifiers (3/4)

- The first part, (\+|-){0,1} matches the preceding sign symbol if there is one
- Note that the minimum number of times is 0
- This part still matches if the sign symbol is absent
- Right after the optional sign symbol are the digits
- We establish that an even number has the least significant digit being 0, 2, 4, 6 or 8

Quantifiers (4/4)

- Therefore, on the far right we specify it as the last digit.
- In between the sign symbols and the least digit there can be zero or more digits.
- Perl defines three special symbols to represent three most commonly used quantifiers
  - * represents {0,}
  - + represents {1,}
  - ? represents {0,1}
- Because + is a quantifier as a result, it has to be escaped in the example pattern above

Character Classes (1/4)

- A character class includes a list of characters where matching of any of these characters result in a match of the character class
- A character class is constructed by placing the characters inside a pair of square brackets

```
my $retval = ($string =~m/^[+-]?[0123456789]*
             [02468]$/);

my $retval = ($string =~m/^[+-]?[0-9]*[02468]$/);
```

- All characters that appear inside the square brackets belong to one character class

Character Classes (2/4)

- Can define multiple ranges in a character class, for example, [a-zA-Z] matches all lowercase and uppercase forms of English alphabets
- **Inside a character class, if we prefix the list of characters with ^, that means any characters that are not listed results in a match**

`[^0-9]` *matches any character which is not numeric*

- Perl also defines some special character classes that contain lists of common character combinations in pattern matching

Character Classes (3/4)

Character Class	Content
\w	Alphanumeric characters and _ ([a-zA-Z0-9_])
\W	Neither alphanumeric characters nor _ ([^a-zA-Z0-9_])
\s	Whitespace characters ([\t\n\r\f])
\S	Non whitespace characters ([^ \t\n\r\f])
\d	Numeric digits ([0-9])
\D	Non numeric digits ([^0-9])

Table 9.2: Special Character Classes in Perl

Character Classes (4/4)

- Finally, our example pattern to match even integers can be simplified as

```
my $retval = ($string =~m/^[+-]?\d*[02468]$/);
```

Regular Expression Operators - m// - Pattern Matching (1/3)

- The m// operator performs pattern matching
- Options for m//
  - The s option treats the string being searched as if it consists of a single line only
  - The i option matches in a case-insensitive manner. Default pattern matching is case sensitive.

◉ The option g which attempts to carry out a global pattern matching on the string

Regular Expression Operators - m// - Pattern Matching (2/3)

- Therefore,

```
'ABCD' =~m/abc/i
```

results in a match

- The **pos()** function – retrieve the position of the current search pointer

Regular Expression Operators - m// - Pattern Matching (3/3)

- We can use this option to find out the position of occurrences of certain patterns in the string

```
$string = 'Telephone: 1234-5678';
while ($string =~m/(\d{4})/g)
{
        print "'$1' found at position " .
        (pos($string) - length($1)) . ".\n";
}
```

- Output is

  '1234' found at position 11.
  '5678' found at position 16.

s///- Search and Replace (1/2)

- The first argument - the search pattern
- The second argument is the replacement string
- The options mentioned above that applies to m// also apply to s///
- The option e causes the replacement string to be treated as an expression instead of a double-quoted string

s///- Search and Replace (2/2)

```
$string =~s/\t/' ' x 4/eg;
# change all tabs to 4 spaces

$string =~s/^(.*)\n$/$1/s;
# like chomp(), to remove trailing newline
```

tr///- Global Character Transliteration

- tr/// changes a set of characters into another
- The first argument is the character list to search for
- The second argument is the character replacement list

```
tr/a-z/A-Z/
```

- Example, convert characters to uppercase