Perl Regular Expressions: Kleene Star

- Kleene star and friends
  - * operator ✍ match zero or more times
    - Precedence is tighter than union ✍ /very* long string/
  - + operator ✍ match one or more times
    - /b+/ same as /bb*/
  - ? operator ✍ match zero or one times
    - /b?/ same as /b|/
  - {NUM} ✍ match exactly NUM times
    - /b{3}/ same as /bbb/
  - {MIN,MAX} ✍ match between MIN and MAX times
    - /b{2,4}/ same as /bb|bbb|bbbb/
  - {MIN,} ✍ match at least MIN times
    - /b{3,} same as /bbbb*/ same as /bbb+/ same as /b{3}b*/

Perl Regular Expressions: Special Characters

- Character class shortcuts
  - \d = [0-9]
  - \s = whitespace (tabs, spaces, newlines)
  - \w = [a-zA-Z0-9_]
  - Negated equivalents
    - \D = [^\d], \S = [^\s], \W = [^\w]
  - Examples
    - /[1-9]?\d/ ✍ one and two digit numbers
    - /\W\w+\d{2,}\W/ ✍words ending in two or more numbers

Perl Regular Expressions: Special Characters

- Other special characters
  - . ✍ match any character
    - /c..?t/ matches “cat”, “cart”, “c&t”, not “chart”
  - ^ ✍ match beginning of string
    - /art/ matches “carts”, /^art/ matches “artistic”, not “carts”
  - $ ✍ match end of string
    - /art$/ matches “chart”, not “artistic”
    - /^art$/ only matches “art”
  - \t,\n ✍ match tab character, newline character
  - \b ✍ match word boundary
    - Similar to /\W\w|\w\W/ (but zero-length)
  - \[|?*/, etc.] ✍ match raw character

Perl Regular Expressions: Variable Interpolation

- Regular expression patterns are treated as interpolated strings
  - Search for a custom word:
    - $searchword = <STDIN>;
      if ($var =~ /$searchword/) {
      print “$searchword has been found.\n”;
      }
  - Make regular expressions readable:
    - Option #1
      - If (/\d+(\.\d+)? [+\-*/%] \d+(\.\d+)?/) {print “Equation\n”;}
    - Option #2
      - $number = ‘\d+(\.\d+)?’;
        $operation = ‘[+\-*/%]’;
        if (/$number $operation $number/) {print “Equation\n”;}

Perl Regular Expressions: Modifiers

- Modifiers: change matching behavior
- Usage: //{modifier}
- Examples:
  - //i ✍ case-insensitive matching
  - //g ✍ match multiple times (global)
    - Not default: not always efficient
  - //m ✍ multiline matching
    - ^ and $ match \n
  - //s ✍ single line matching
    - . Matches newline

Perl Regular Expressions: Substitution

- Purpose:
  - Replace strings with other strings (“search and replace”)
- Usage:
  - s/searchpattern/replacepattern/;
  - s/searchpattern/replacepattern/g;
  - s[searchpattern][replacepattern], etc.
  - $variable =~ s/search/replace/;
- Returns:
  - Number of successful replacements
  - “” if no replacements (same as 0)

Perl Regular Expressions: Substitution

- Substitution examples:
  - Grammar check: eliminate “the the”
    - foreach (@list) {
      s/the the/the/g;
      }
  - Eliminate comments from a Perl file:
    - while (<INFILE>) {
      s/#.*//;
      print;
      }

Perl Regular Expressions: Matching

- Matching examples:
  - Skip blank lines in a file:
    - while (<INFILE>) {
      next if /^\s*$/;
      processContents();
      }
  - Only print numbers from a list:
    - foreach (@mixedlist) {
      print if /^\s*\d*(\.\d+)?\s*$/;
      }

Perl Regular Expressions: Translation

- Purpose:
  - Replace all instances of one character with another character
- Usage:
  - tr/searchlist/replacelist/;
  - tr:searchlist:replacelist:, etc.
  - $variable =~ tr/searchlist/replacelist/;
- Returns:
  - Number of characters replaced

Perl Regular Expressions: Translation

- Translation examples:
  - Capitalize all lower-case letters:
    - tr/a-z/A-Z/;
  - “Invert” all numbers in a string:
    - $number =~ tr/0-9/9876543210/;
  - Print words from a file line-by-line:
    - while (<INFILE>) {
      tr/ /\n/;
      print;
      }

Perl Regular Expressions: Translation

- Translation Nuances
  - If the replacement list is too short:
    - Final search list character is replicated
      - tr/a-zA-Z/b/; ✍ replace all alphabetic letters with ‘b’
    - d modifier: “missing” characters are deleted
      - tr/a-zA-Z/b/d; ✍ eliminates all letters except ‘a’
  - If the replacement list is empty:
    - No replacements are made
    - Useful for counting characters
  - Translation does not interpolate

Back References

- Scenarios:
  - Find all word duplicates in a sentence
  - Switch the position of two words
  - Multiply all numbers by 1 million
- Back References
  - Identify submatches
  - Manipulate submatches

Back References

- Identifying Submatches
  - Group matches with parentheses
    - Find two numbers in a string: /\d+\s\d+/
    - Identify each number: /(\d+)\s(\d+)/
    - Be careful: /(dog|cat)/, /(\d)+/
  - Identify matches with \1, \2, \3 …
    - Order is determined from left to right
      - Same for nested parentheses: /((\d+)\s(\d+))\s(\w+(\d))/
    - Find the same number twice: /(\d+)\s\1/
    - Find a vowel-consonant pair repeated twice:
      - /([aeiou])([^aeiou])\1\2/

Back References

- Long-term Identification
  - \1, \2, \3… ✍ $1, $2, $3 …
  - Variables last until
    - New regular expression match
    - Current scope ends
  - Example:
    - while ($a = <STDIN>) {
      if ($a =~ /(\w+)\s(\w+)/) {
      print “Words reversed: $2 $1\n”;
      }
      }

Back References

- Matching and Back References
  - Returns
    - Scalar context: 1 on success, 0 on failure
    - List context: ($1, $2, $3…)
      - Null list if no matches
      - (1) if no capturing parentheses
- Substitution and Back References
  - s/regexp/replace/;
    - Left side uses \1, \2, \3, …
    - Right side uses $1, $2, $3…
    - Eliminate the last letter of a word: s/(\w+)\w\b/$1/;

RE Evaluation Techniques

- Evaluation Rule: greedy evaluation
  - For each operation, match as much as possible
    - $a = “aaaab”;
      $a =~ /a*/;
  - Think about the short term (even at the expense of the long term)
    - $a = “acaaabbb”;
      $a =~ /a*b*/;

Perl References

- O’Reilly – Programming Perl by Wall
- www.perlmonks.com
- www.activestate.com - Komodo IDE