CS 3101-4 - Programming Languages: Perl

Lecture 3: Scopes, Pattern matching

Vinodkumar Prabhakaran
vinod@cs.columbia.edu

April 5, 2012

Contents

Homework 1

- questions, remarks?
- problem #1 - map, foreach
- problem #2 - fine, except for some silly mistakes
- problem #3 - missed out the NIL
- problem #4
  - decreasing vs increasing fine
  - no sorting
  - $count vs scalar(keys %hash)

Homework 2

- to be posted tonight
- submit via courseworks
- by Wednesday, April 11th, night 11.59 EST

Project Proposal

- proposal due on April 9th. (submit early to start early)
- groups of two allowed, but grading criteria would be higher

Summary

- Input/Output
  - open, close, read (< ... >), write (print)
  - while(<>) ...
- More on arrays, hashes, lists
  - arrays: push, pop, shift, unshift, delete
  - arrays: split, join, sort, reverse, map, grep
  - hashes: sorted order
  - hashes: keys, values, each, delete, reverse
  - hashes: exists, defined
- Subroutines
  - defining (anywhere) and invoking
  - return values, implicit and explicit
  - passing args @_, arbitrary number of args

A better max (from last week)

Works for arbitrary number of arguments!

```
1  sub max {
2    my $max_so_far = shift;   # first is the max so far
3    foreach (@_) {            # look at rest
4      if ($_ > $max_so_far) {
5        $max_so_far = $_;
6      }
7    }
8    $max_so_far;
9  }
10 print &max (12,231,34,245,6,46); # 245
11 print &max (34,142,15); # 142
```

my vs local vs our

`my`

- creates a private variable visible only within the block
- invisible outside the block
- lexical scope

`local`

- saves the variable's current value in stack
- temporary values assigned to it within the current scope
- variable is still **global**!
- dynamic scope

`our`

- gives access to global variables defined in other packages!
- more on it when we look at packages.

my vs local: example

```
1  $office = "global";   # Global $office
2
3  &say(); # says "global"
4  &barney(); # says "barney global", lexical scope;
5  &fred();  # says "fred fred", dynamic scope,
6  &say(); # says "global", restored after &fred()
7
8  sub say { print "$office\n"; }  # print the $office
9
10 sub barney { my $office = "barney";
11  print "$office "; &say(); }
12
13 sub fred { local $office = "fred";
14 print "$office "; &say(); }
```

Pragmas

- compiler directives to ensure discipline
- not executed, but tells compiler how lenient it should be

```
1  use warnings;
2  # Enable warnings till end of file
3  ...
4  {
5      no warnings;
6      # Disable warnings till end of block
7      ...
8  }
9  # warnings are enabled back
```

Pragmas

```
1 use warnings;
```

- Perl complains about variables that are used only once, variable re-declarations, improper conversions etc.

```
1 use strict;
```

- Perl throws compilation error if the variable is
  - not declared using `my` (lexical) or `our` (global)
  - not one of special variables ($_, $"...)
  - not qualified with a package name (e.g. $Foo::var)
  - not imported into current package using `use` or `require`

An example: find all course number mentions

MS degree requirement

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).
1 required course: COMS W4236.
1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

List all course mentions

Have an array of all course numbers and check if its in each line?

```
@allcourses=("COMS W3000",...,"CSOR W3000",...);
while(<>) {
 foreach $x (@allcourses) {
  if (index($_,$x)>0) { # returns index of $x in $_
       print $x, ", ";
  }
 }
}
```

Pattern matching

- to find patterns in text without listing all possible instances
- Perl pattern matching operators
  - Match: m/PATTERN/ (m optional)
  - Substitute: s/PATTERN/REPLACEMENT/
- bind the matching/substituting to a variable
  - $line =~ m/PATTERN/
  - $line !~ m/PATTERN/
- Regular expression: a 'language' to specify patterns

An example: find all course number mentions

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).
1 required course: COMS W4236.
1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

```
while(<>) {
 if($_ =~ m/COMS W\d\d\d\d/) {
        print $&, ", "; #$& contains the matched part
  }
}
```

```
COMS W3261, COMS W4236, COMS W4203,
```

Quantifiers

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).
1 required course: COMS W4236.
1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

```
while(<>) {
 if($_ =~ m/COMS W\d{4}/) {
        print $&, ", ";
  }
}
```

```
COMS W3261, COMS W4236, COMS W4203,
```

Alternators

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).
1 required course: COMS W4236.
1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

```
while(<>) {
 if($_ =~ m/(COMS|CSOR) W\d{4}/) {
        print $&, ", ";
  }
}
```

```
CSOR W4231, COMS W4236, COMS W4203,
```

Modifiers

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).
1 required course: COMS W4236.
1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

```
while(<>) {
  while($_ =~ m/(COMS|CSOR) W\d{4}/g) {
        print $&, ", ";
  }
}
```

```
CSOR W4231, COMS W3261, COMS W4236, COMS W4203, COMS W
    4205, COMS W4241, COMS W4252, COMS W4261, COMS W
    4281
```

Character classes

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).
1 required course: COMS W4236.
1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

```
while(<>) {
  while($_ =~ m/[A-Z]{4} [A-Z]\d{4}/g) {
        print $&, ", ";
  }
}
```

```
CSOR W4231, COMS W3261, COMS W4236, COMS W4203, COMS W
    4205, COMS W4241, COMS W4252, COMS W4261, COMS W
    4281
```

Grouping

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).
1 required course: COMS W4236.
1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

```
while(<>) {
  while($_ =~ m/([A-Z]{4} [A-Z]\d{4})/g) {
        print $1, ", ";
  }
}
```

```
CSOR W4231, COMS W3261, COMS W4236, COMS W4203, COMS W
    4205, COMS W4241, COMS W4252, COMS W4261, COMS W
    4281
```

Grouping

Fulfill the 12-credit core requirement. One of the core requirements must be CSOR W4231. In addition, COMS W3261 or past equivalent is a required pre-requisite (No MS credit for W3261).
1 required course: COMS W4236.
1 course chosen from the "Electives I" list: COMS W4203, COMS W4205, COMS W4241, COMS W4252, COMS W4261, or COMS W4281.

```
while(<>) {
  while($_ =~ m/(([A-Z]{4}) (([A-Z])(\d{4})))/g) {
        print $5, ", ";
  }
}
```

```
4231, 3261, 4236, 4203, 4205, 4241, 4252, 4261, 4281,
```

Quantifiers

- to specify the number of times a pattern (or sub-pattern) should repeat
- * (zero or more), + (one or more), ? (zero or one), {n}, {n,}, {n,m}

```
1 if ($line =~ /abc/) # abc
2 if ($line =~ /abc*/)  # ab, abccc, but not abcabc
3 if ($line =~ /abc+/)  # abccc, but not ab or abcabc
4 if ($line =~ /(abc)+/)  # abc, abcabc etc.
5 if ($line =~ /abc?/)  # ab and abc
6 if ($line =~ /abc{2}/)  # abcc and nothing else
7 if ($line =~ /(abc){2}/)  # abcabc and nothing else
8 if ($line =~ /abc{2,}/)  # abcc, abccc and so on
9 if ($line =~ /abc{2,4}/)  # abcc, abccc or abcccc
```

Magic .

. matches any character (wild card)

```
1 if ($line =~ /<xml>(.*)<\/xml>/)
2 # match any text in between two <xml> xml tags
```

Greedy matching

```
1 $line = "Want this <bold>content's tag<\/bold> matched
    ";
2 if ($line =~ /<(.*)>/) # matches "bold>content's tag
    <\/bold"
3 if ($line =~ /<(.*?)>/) # matches "bold"
4 if ($line =~ /<(.+?)>/) # matches only if its non
    empty
```

Two different uses of ?: a? vs. a*?

Modifiers and character classes

- modifier g - global match
- modifier i - ignore CaSe

```
1 if ($line =~ /COMS W\d{4}/i)
2 # matches "coms 3101" as well as "COMS 3101" or even "
      CoMs 3101"
3 while ($line =~ /COMS W\d{4}/gi) { ... }
```

- character class [abcd] = a|b|c|d
- can specify ranges [a-z], [A-Z], [a-zA-Z0-9]
- negate entire character class [^a-z]

```
1 # MTA bus names
2 if ($line =~ /[MBQ] \d{1,3}/) # M 104, M 60, Q 10
3 # Full names with optional initials
4 if ($line =~ /[A-Z][a-z]+ ([A-Z]\.? )?[A-Z][a-z]+/)
```

Special character classes

- \t, \n, \r
- \w, \W (word/non-word characters) [A-Za-z_]
- \s, \S (space/non-space characters) (space/tab/newline)
- \d, \D (digit/non-digit characters) [0-9]

```
1 if ($line =~ /ab\tc/) # ab  c
2 if ($line =~ /\w+:\w+/)   # x4_z:7dhf
3 if ($line =~ /\w+\W\w+/)   # x4_z:7dhf, x4_z#7dhf
4 if ($line =~ /ab\s*c/)   # abc, ab  c, ab    c
5 if ($line =~ /[a-z]{2,3}\d{4}/)   # Columbia UNI?
```

Referring back to groups

- Within the pattern: \N denotes the N^{th} group from the beginning

```
1 if ($line =~ /<(.*?)>.*?<\/\1>/)
2 # matches any tags <xyz>.....</xyz>
```

- After the pattern: $N denotes the N^{th} group from the beginning

```
1 if ($line =~ /<(.*?)>(.*?)<\/\1>/) {
2     print "$1 : $2";
3 }
```

Assersions (zero width characters)

- ^ and $ assert beginning and end of string
- \b match at word boundary
- \B match NOT at word boundary

```
1 if ($line =~ /^S\d{4}:/) # Any line starting with S
    followed by 4 digits
2 if ($line =~ /\?$/ # Any line ending in a ?
3 if ($line =~ /^Why (.*)[^!?]$/)  # Any line that
    starts with a 'Why' but does NOT end in a ? or !
```

Dirty dozen

Should escape within the pattern using \

- \- next character has special meaning
- | - alternator
- (- grouping
-) - grouping
- [- character class
- { - quantifier
- ^ - match at the beginning
- $ - match at the end
- * - zero or more times
- + - one or more times
- ? - optional (zero or one time)
- . - match anything

Substitution

- s/PATTERN/REPLACEMENT/
- substitutes PATTERNs with REPLACEMENT

```
1 $line =~ s/foo/boo/; # foofoo ==> booboo
2 $line = "<text>this is the text</text>";
3 $line =~ s/^<(.*?)>(.*?)<\/\1>$/$1:$2/;
4 print $line; # text:this is the text
```

Return values

- m//
  - scalar context - returns true (1) if successful, else false ("")
  - list context - returns a list of matched groups
- s///
  - returns number of times it succeeded (scalar & list context)

```
1  $line = "<text>this is the text</text>";
2  ($tag,$content) = $line =~ /^<(.*?)>(.*?)<\/\1>$/;
3
4  if (@perls = $text =~ /perl/gi) {
5    print "Number of times Perl mentioned : ", scalar(
         @perls);
6  }
7
8  $string = "name=xyzzy id=9 score=0";
9  %hash = $string =~ /(\w+)=(\w+)/g;
10
11 $num = $text =~ s/perl/PERL/g;
```

Pattern matching in action

Skip all comment lines

```
1 while(<>) {
2   next if $line =~ /^#/;
3   ...
4 }
```

Split a file path

```
1 $path = "/user/programs/hw1/words.txt";
2 ($dir,$base,$dum,$ext) = $path =~ /^(.*\/)(.*?)
      (\.(.*?))?$/;
3 print "Dir=$dir, Base=$base, Ext=$ext\n";
```

```
1 Dir=/user/programs/hw1/, Base=words, Ext=txt
```

Pattern matching in action

Build a hash

```
1 from:  vp2198@columbia.edu
2 to:  vp2198@columbia.edu
3 date:  Sun, Apr 1, 2012 at 11:08 PM
4 subject:  COMS W3101.004 Office Hours Change (this
      week only)
5 mailed-by:  columbia.edu
```

```
1 while(<>) {
2     $_ =~ /^(.*?):\s*(.*)$/;
3     $hash{$1} = $2;
4 }
```

Back tick - `...`

```
1 $num_files = `ls | wc -l`;
2 @files = `ls`; # ("file1", "file2",...)
3 $files = `ls`; # "file1\n$file2\n..."
```

Filters on lists

Filters: operators transform lists to other lists (e.g. `sort`)

You can apply list filters sequentially

```
1 print reverse sort keys %agehash;
2
3 # you can think of it as a long nexted function call!
4 print (reverse (sort (keys %agehash)));
```

Act like unix shell filters

```
1 # shell commands (NOT PERL)
2 cat article.txt | tr ' ' '\n' >words.txt
3 cat words.txt | grep 'ing$' | sort | uniq -c | sort
```

Special variables

- $_ - default input (and few other things)
- $. - input line number
- $/ - input record delimiter
- $\- output record separator
- $, - output field separator
- $" - similar to $,. in array interpolation
- $! - error number or message
- $0 - name of the program/script file
- $1... - matched groups in PATTERN
- $&, $`, $' - matched, preceding, following
- ▸ Link

Summary

- Leftovers
  - scope declarations: `my` vs `local`
  - pragmas
  - backticks, list filters, special variables
- Pattern matching
  - matching and substituting
  - quantifiers, alternations, modifiers, grouping, character classes
  - greedy vs. non-greedy matching
  - referring to groups within the pattern and outside
  - http://files.meetup.com/120908/Regexp Quick Reference.pdf
  - ▸ Link