Script Programming with Perl

15-123

Systems Skills in C and Unix

Scripting Languages

- Many routine programming tasks require custom designed solutions, environments and approaches
  - Extracting data from a roster file
- Scripting languages are ideal for tasks that do not require a “high level” compiled language solution
  - Some argue that this is the real way to learn programming
  - No need to worry about static typing
- Scripts are widely used as backend processing languages for web based applications
  - Authenticate passwords
  - Extract data from a database
  - Create dynamic web pages

Popular Scripting Languages

- JavaScript
  - Client side processing based on a built in browser interpreter
- PHP
  - Server side processing
- Python
  - Object oriented, interpreted, data structures, dynamic typing, dynamic binding, rapid application development, binding other programming components
- Perl
  - Also you can call it an “*interpreted*” language (more later)

Perl

- An interpreted scripting language
  - Practical extraction and Report Language
  - Developed as a tool for easy text manipulation and report generation
- Why Perl
  - Easy scripting with strings and regex
  - Files and Processes
- Standard on Unix
- Free download for other platforms

What’s good for Perl?

- Scripting common tasks
- Tasks that are too heavy for the shell
- Too complicated (or short lived) for C

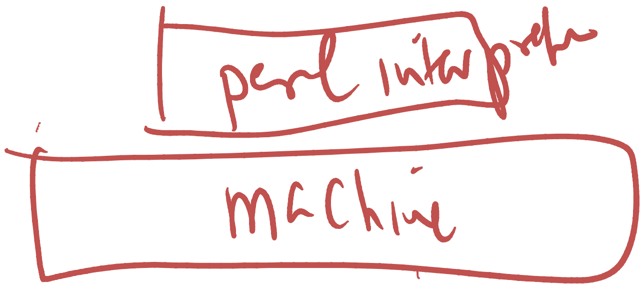
$$\begin{tikzcd}
\text{shell} \arrow[r, dash] & \text{perl} \arrow[r, dash] & \text{C} \\
\text{short term} \arrow[u] & & \text{long term} \arrow[u]
\end{tikzcd}$$

First Perl Program

```
#!  usr/bin/perl  –w
print (“hello world \n”);
```

first.pl

- How does this work?
  - Load the interpreter and Execute the program
    - perl hello.pl



An interpreted language

- Program instructions do not get converted to machine instructions.
- Instead program instructions are executed by an “interpreter” or program translator
- Some languages can have compiled and interpreted versions
  - LISP, BASIC, Python
- Other interpreters
  - Java interpreter (byte code) and .net CIL
    - Generates just in time machine code

Perl Data Types

- Naming Variables
  - Names consists of numbers, letters and underscores
  - Names cannot start with a number
- Primitives
  - Scalars
    - Numeric : 10, 450.56
    - Strings
      - ‘hello there\n’
      - “hello there\n”

Perl Data Types

- arrays of scalars
  - ordered lists of scalars indexed by number, starting with 0 or with negative subscripts counting from the end.
- associative arrays of scalars, a.k.a``hashes''.
  - unordered collections of scalar values indexed by their associated string key.

Variables

- $a = 1; $b = 2;
- All C type operations can be applied
  - $c = $a + $b; ++$c; $a +=1;
  - $a ** $b - something new?
- For strings
  - $s1 . $s2   - concatenation
  - $s1 x $s2 - duplication
- $a = $b
  - Makes a copy of $b and assigns to $a

Useful operations

- **substr($s, start, length)**
  - substring of $s beginning from **start** position of **length**
- **index string, substring, position**

  look for first index of the substring in string starting from position
- **index string, substring**

  look for first index of the substring in string starting from the beginning
- **rindex string, substring**

  position of substring in string starting from the end of the string
- **length(string) –** returns the length of the string

More operations

- **$_ = $string; tr/a/z/;   # tr is the transliteration operator**

  replaces all ‘a’ characters of string with a ‘z’ character and assign to $1.
- **$_ = $string; tr/ab/xz/;**

  replaces all ‘a’ characters of string with a ‘x’ character and b with z and assign to $1.
- **$_ = $string; s/foo/me/;**

  replaces all strings of “foo” with string “me”
- **chop**

  this removes the last character at the end of a scalar.
- **chomp**

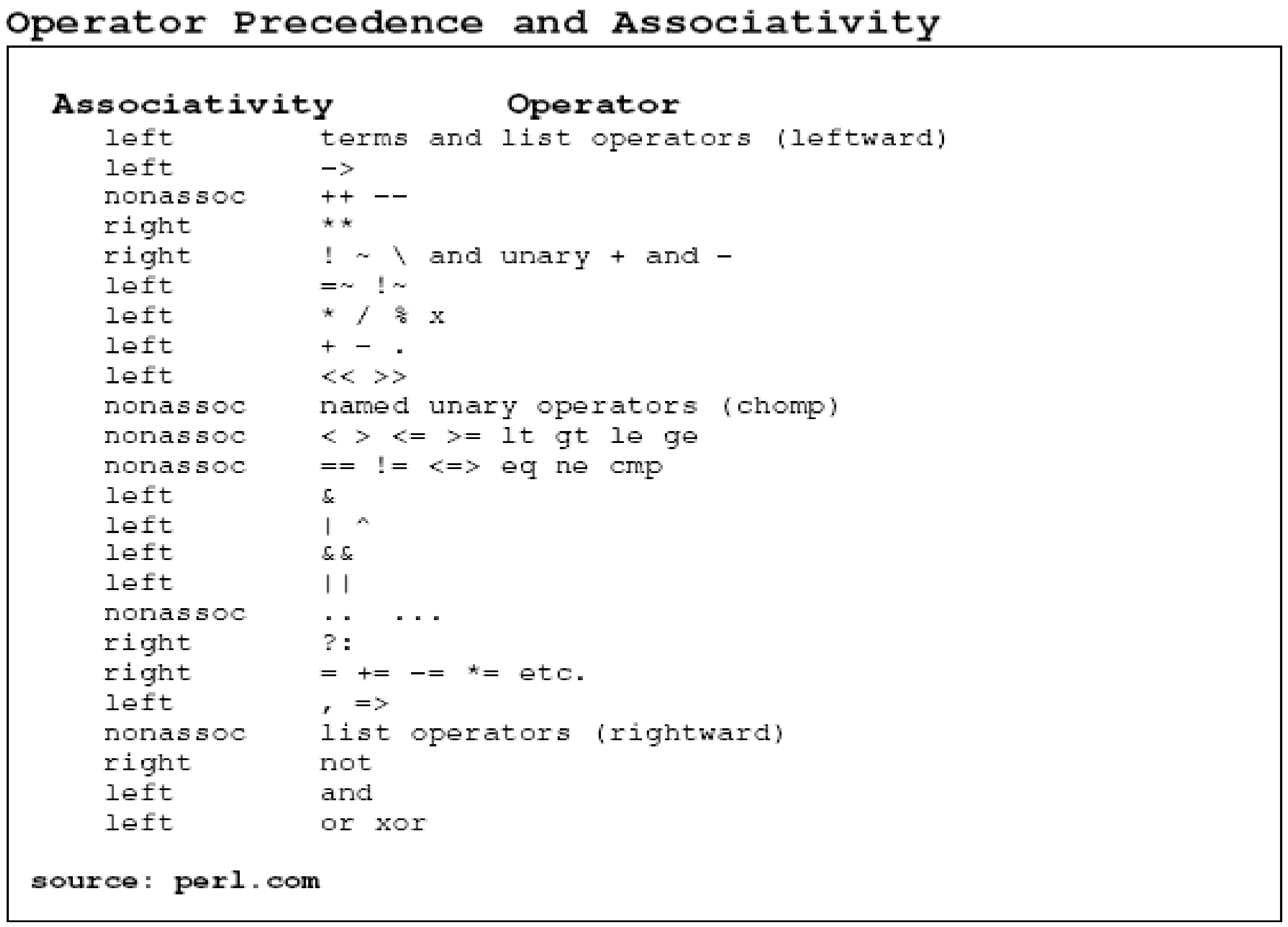
  removes a newline character from the end of a string
- **split splits a string and places in an array**

o @array = split(/:/,$name); # splits the string $name at each : and stores in an array

o **The ASCII value of a character $a is given by ord($a)**

Comparison Operators

Comparison	Numeric	String
Equal	==	Eq
Not Equal	!=	Ne
Greater than	>	Gt
Less than	<	Lt
Greater or equal	>=	Ge
Less or equal	<=	Le



Operator Precedence and Associativity

Associativity	Operator
left	terms and list operators (leftward)
left	->
nonassoc	++ --
right	**
right	! ~ \ and unary + and -
left	=~ !~
left	* / % x
left	+ - .
left	<< >>
nonassoc	named unary operators (chomp)
nonassoc	< > <= >= lt gt le ge
nonassoc	== != <=> eq ne cmp
left	&
left	| ^
left	&&
left	||
nonassoc	.. ...
right	?:
right	= += -= *= etc.
left	, =>
nonassoc	list operators (rightward)
right	not
left	and
left	or xor

source: perl.com

More at: http://www.perl.com/doc/manual/html/pod/perlop.html

Arrays

$A[$i]

- @array = (10,12,45);
- @A = (‘guna’, ‘me’, ‘cmu’, ‘pgh’);
- Length of an array
  - $len = $#A + 1
- Resizing an array
  - $len = desired size

@A = (1,2,3);

$#A=5

$\boxed{\$\#A = 1} \rightarrow (1,2)$

repetition

A While Loop

```
$x = 1;
while ($x < 10){
  print "x is $x\n";
  $x++;
•   }
```

Until loop

```
$x = 1;
until ($x >= 10){
print "x is $x\n";
$x++;
}
```

repetition

Do-while loop

```
$x = 1;
do{
    print "x is $x\n";
    $x++;
} while ($x < 10);
```

for statement

```
for ($x=1; $x < 10; $x++){
   print “x is $x\n”;
}
```

foreach statement

```
foreach $x (1..9) {
    print "x is $x\n";
}
```

Parsing a roster entry

$S =

- 'S10,guna,Gunawardena,Ananda,SCS,CS,3,L,4, 15123 ,A ,,'

```
@array = split(',', $S);

printf "%s \n", $array[3];
```

Perl IO

< file.txt

> file.txt

>> file.txt

```
$size = 10;
open(INFILE, “file.txt”);
$#arr = $size-1; # initialize the size of the array to 10
$i = 0;
foreach $line (<INFILE>) {
  $arr[$i++] = $line;
  if ($i >= $size) {
     $#arr = 2*$#arr + 1; # double the size
     $size = $#arr + 1;
  }
}
```

Perl IO

- open(OUT, “>out.txt”);
- print OUT “hello there\n”;
- Better file open
  - open (OUT, “>out.txt”) || die “sorry out.txt could not be opened\n”

Perl and Regex

Perl and Regex

- Perl programs are perfect for regex matching examples
  - Processing html files
    - Read any html file and create a new one that contains only the outward links
    - Do the previous exercise with links that contain cnn.com only

Regex syntax summary

- ?, +, *
- (   )   - grouping
- (exp (exp)) ➔ \1, \2 or $1 , $2 backreference matching
- ^startwith
- [^exclusion group]
- [a-z,A-Z] – alpha characters

Perl and regex

```
open(INFILE, "index.html");
foreach $line (<INFILE>) {
  if ($line =~ /guna/ ){
      print $line;
   }
}
close(INFILE);
```

=~ binding

=! exclusion

Lazy matching and backreference

```
open(IN, “guna.htm”);
while (<IN>){
  if ($_ =~ /mailto:(.*?)"/){
     print $1."\n";
  }
}
```

back reference

"mailto: guna@cs.cmu.edu"

Global Matching

- How to find all matches on the same line

```
open(IN, “guna.htm”);
while (<IN>){
   if ($_ =~ /mailto:(.*?)"/g){
      print $1."\n";
   }
}
```

Global Matching and Replacing

The statement

```
$str =~ s/oo/u/;
```

would convert "Cookbook" into "Cukbook", while the statement

```
$str =~ s/oo/u/g;
```

would convert "Cookbook" into "Cukbuk".

CGI Scripts and Perl

- CGI is an interface for connecting application software with web servers
- CGI scripts can be written in Perl and resides in CGI-bin
- Example: Passwd authentication

```
while (<passwdfile>) {
  ($user, $passwd)= split (/:/, $_);
  ............
}
```

LWP
Library for www in Perl

- LWP contains a collection of Perl modules
  - ***use LWP::Simple;***
  - ***$_ = get($url);***
  - ***print $_;***
- *Good reference at*
  - *http://www.perl.com/pub/a/2002/08/20/perlandlwp.html*

Getopt

- The Getopt::Long module implements an extended getopt function called GetOptions().
- Command line arguments are given as
  - **-n 20   or –num 20**
  - **-n 20 -t test**

- ***use Getopt::Long;***
- ***$images_to_get = 20;***
- ***$directory = ".";***
- ***GetOptions("n=i" => \$images_to_get, "t=s" => \$directory);***

References: http://perldoc.perl.org/Getopt/Long.html