CS 3101-4 - Programming Languages: Perl

Lecture 4: References, Data structures

Vinodkumar Prabhakaran
vinod@cs.columbia.edu

April 12, 2012

Contents

Announcements

Homework 2

- grades will be posted over the weeekend

Project Proposal

- proposal reviews sent
- submit revised proposals by tonight, if asked

Homework 3

- to be posted tonight
- submit via courseworks
- by Wednesday, April 18th, 11.59 pm EST

Review

- Scope declarations: `my` vs `local`
- Pragmas: `use warnings`, `no warnings`, `use strict`
- 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

References for References

- Perl References Tutorial ▸ Link
- Perl Lists of Lists ▸ Link
- Perl Data Structures Cookbook ▸ Link

Restaurants and their neighborhoods

- Restaurant => neighborhood

```
%hash = (
   "Dangi"=>"HK",
   "Sigiri"=>"EV",
   "Takashi"=>"WV",
   "Taboon"=>"HK",
   "Casa"=>"WV"
);
print "$rest is in $hash{$rest}";
```

Neighborhoods and their restaurants

- How to represent neighborhood => restaurants?
- Arrays and hashes can only have scalars as values
- We could merge all restaurants in one hood into one string

```
%hash = (
   "HK" => "Dangi,Taboon",
   "EV" => "Sigiri",
   "WV" => "Takashi,Casa",
);

for $hood (keys %hash) {
   print "List of restaurants in $hood";
   @restaurants = split(/,/, $hood);
   print "$_\n" foreach (@restaurants);
}
```

Neighborhoods and their restaurants

```
%hash = (
   "HK" => "Dangi,Taboon",
   "EV" => "Sigiri",
   "WV" => "Takashi,Casa",
);

for $hood (keys %hash) {
   print "List of restaurants in $hood";
   @restaurants = split(/,/, $hood);
   print "$_\n" foreach (@restaurants);
}
```

- Messy way to maintain
- How about accessing 5th restaurant in HK?
- What if we have values that may contain ‘,’?
- Wish there were hashes of arrays?

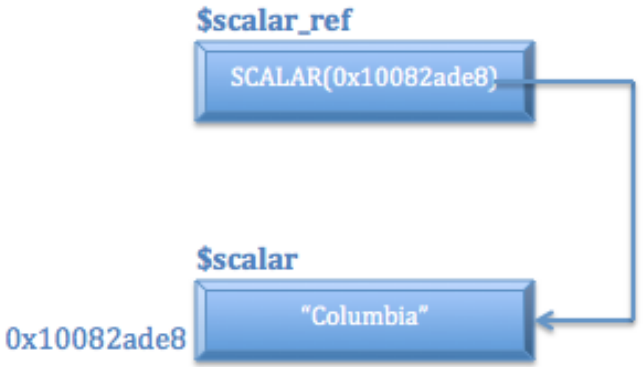
References

- yet another type of scalar (other types: string, number etc.)
- holds the reference to the location where any type of variable resides: scalar, array or hash
- similar to C pointers, but different.
  - can refer to data as well as procedures (we'll see later)
  - but don't let you peek and poke at raw memory locations
- if you have a reference to an array/hash, you can access entire array/hash using it, but reference is just a scalar.
- you can make hash of references to arrays (simply called, hash of arrays)! Yay!

References to other variables

- '\' gets the reference of a variable
- refers to location of a variable

```
$scalar_ref = \$scalar;
```



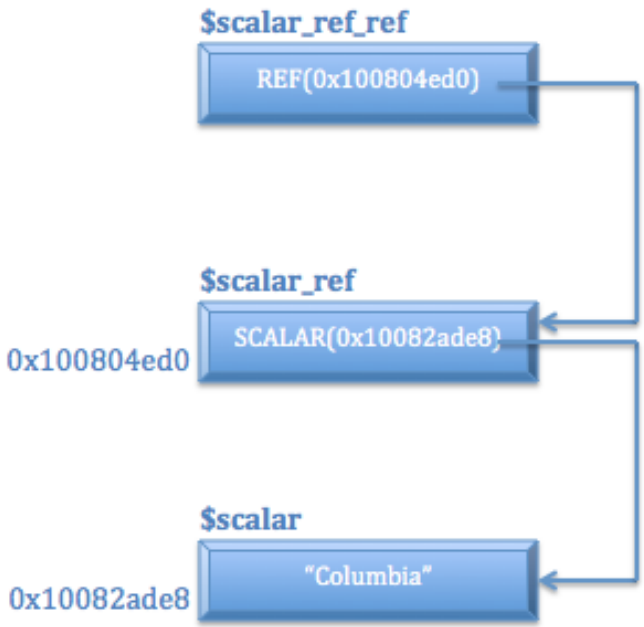
References to other variables

- '\' gets the reference of a variable
- refers to location of a variable

```
$scalar_ref = \$scalar;
```

- can do reference of reference and so on

```
$scalar_ref_ref = \$scalar_ref;
```



Getting references to variables

```
$a_scalar = "boom";
@an_array = ("Columbia", 2012, "Spring");
%a_hash = ("Dangi"=>"HK","Sigiri"=>"EV");

$scalar_ref = \$a_scalar;
$array_ref = \@an_array;
$hash_ref = \%a_hash;

# They're just scalars,
# They can be used like any other scalar

$one_ref = $another_ref;
$arr[3] = $a_ref;
$h{$i} = $a_ref;
```

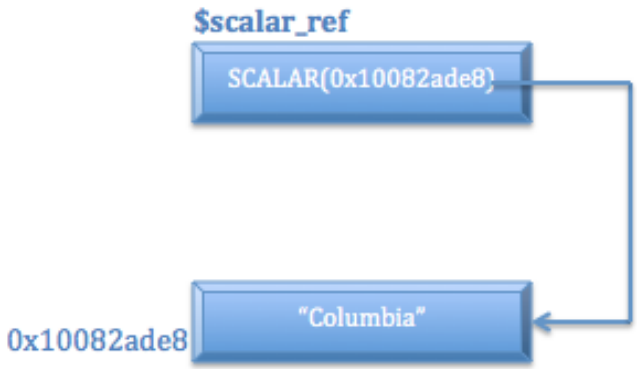
Getting references to variables

```
@refs = \($one,$two,$three);
# equivalent to
@refs = (\$one,\$two,\$three);

@refs = \($the_scalar, @the_array, %the_hash);
# equivalent to
@refs = (\$the_scalar, \@the_array, \%the_hash);

@refs = \(@x) # interpolate and then get refs
# equivalent to
@refs = map { \$_ } @x;

# What about?
@refs = \(@x, (@y));
```

Anonymous references

- Do we really always need a variable name for indirection?
- No, you can create anonymous references

```
$scalar_ref = \"Columbia";
```



Creating anonymous references

```
$scalar_ref = \"Columbia";
$another_scalar_ref = \456;

# how about arrays?
$array_ref = \(43,23,13); # wrong!
$array_ref = [43, 23, 13];
#square brackets put to different use than $arr[3]

# how about hashes?
$hash_ref = {APR => 4, MAY => 5};
#curlies put to different use than $hash{$key}

# equivalent to
%a_hash = (APR => 4, MAY => 5);
$hash_ref = \%a_hash;
# same for arrays
```

Using references with curlies

```
print ${$scalar_ref}; # prints "Columbia"
${$scalar_ref} = "Yale"; # prints "Yale", if we print

@bkp_array = @{$array_ref};
# bkp_array now has (43,23,13);
${$array_ref}[2] = 93;
# $array_ref now refers to array (43,23,93)

%bkp_hash = %{$hash_ref};
# bkp_hash now has (APR,4,MAY,5);
${$hash_ref}{MAY} = 99;
# $hash_ref now refers to hash with MAY => 99
```

Do we have to live with curlies?

- curlies can be omitted when the thing inside them is an atomic scalar variable (like $hash_ref, but not $a_ref[0])

```
$$scalar_ref = "Yale"; # prints "Yale", if we print
@bkp_array = @$array_ref;
$$array_ref[2] = 93;
%bkp_hash = %$hash_ref;
$$hash_ref{MAY} = 99;

$$array_ref[2] = 93;   # ${$array_ref}[2] = 93;
# is NOT same as
${$array_ref[2]} = 93;
```

Using references with arrows

```
${$array_ref}[2] = 93;
$$array_ref[2] = 93;
$array_ref->[2] = 93;

# Similarly
${$hash_ref}{MAY} = 99;
$$hash_ref{MAY} = 99;
$hash_ref->{MAY} = 99;

print $array_ref->[2];
# 3rd item in an array referenced by $array_ref
print $array_ref[2];
# 3rd item in the array @array_ref
```

Multidimensional arrays (kind of)

```
@a = (
   [14,23,3],
   [32,2,13],
   [31,22,2]
);

print $a[2]->[1]; # prints 22
# ${$a[2]}[1] would have worked too, but messy

# $a[ROW]->[COLUMN]
```

Arrows can be skipped between brackets (and braces too)

```
print $a[2][1]; # prints 22
```

Revisiting the restaurant neighborhood problem

Put the restaurants in a hash of references to arrays

```
%hash = (
   "HK" => [Dangi,Taboon],
   "EV" => [Sigiri],
   "WV" => [Takashi,Casa],
); # this is just a regular hash, not a reference

for $hood in (keys %hash) {
   print "List of restaurants in $hood\n";
   print "$_\n" foreach (@{$hash{$hood}});
}
```

Arrays and hashes of arrays

Array of arrays

```
@a = (
   [14,23,3],
   [32,2,13],
   [31,22,2]
);
$a[0][2];
```

Hash of arrays

```
%hash = (
   "HK" => [Dangi,Taboon],
   "EV" => [Sigiri],
   "WV" => [Takashi,Casa],
);
$hash{HK}[0];
```

Building arrays and hashes of arrays

Array of arrays

```
# 35 25 25 24
# 33 22 24 25
# 34 24 24 21
while ( <> ) {
  push @hw_marks,[split]; # an anonym array per line
  }
```

Hash of arrays

```
# HK : Dangi Taboon
# EV : Sigiri
# WV : Takashi Casa
while ( <> ) {
  next unless s/^(.*?):\s*//; # skip other lines
  $hash{$1} = [ split ]; # split $_ on / /
}
```

Hashes of hashes

```
%course_hash = (
  python => {
    instructor      => "Daniel",
    room            => MUDD255
  },
  perl   => {
    instructor      => "Vinod",
    room            => MUDD834
          }
  ); # a regular hash

print $course_hash{python}{instructor}; # Daniel
print $course_hash{perl}{room}; # MUDD834
```

Hashes of hashes

Building a hash of hashes from a file

```
# Input file contains:
# python : instructor=Daniel room=MUDD255
# perl : instructor=Vinod room=MUDD834

while ( <> ) {
  next unless s/^(.*?):\s*//;
  $course = $1;
  for $field ( split ) { # split $_ on / /
    ($key, $value) = split /=/, $field;
    $course_hash{$course}{$key} = $value;
  }
}
print $course_hash{python}{instructor}; # Daniel
print $course_hash{perl}{room}; # MUDD834
```

More complex structures

```
%course_hash = (
  python => {
    instructor      => "Daniel",
    room            => MUDD255
    students        => [Dillen,Elliot,Sarah]
  },
  perl   => {
    instructor      => "Vinod",
    room            => MUDD834
    students        => [Bill,Mehmet,Jenee]
  }
);
print $course_hash{python}{instructor}; # Daniel
print $course_hash{perl}{students}[0]; # Bill
```

Common mistakes

```
for $i (1..10) {
  @array = somefunc($i);
  $array_of_arrays[$i] = @array;   #WRONG
}
for $i (1..10) {
  @array = somefunc($i);
  $array_of_arrays[$i] = \@array;   #WRONG
}
```

```
for $i (1..10) {
  my @array = somefunc($i);
  $array_of_arrays[$i] = \@array;   #OK, thanks to 'my'
}
for $i (1..10) {
  @array = somefunc($i);
  $array_of_arrays[$i] = [@array]; # GOOD
  # $array_of_arrays[$i] = [ somefunc($i)]; # BETTER
}
```

References to subroutines

- Can have references to subroutines too

```
sub max {
# compute max
}
$sub_ref = \&max;
...
```

- Even anonymous ones

```
$sub_ref = sub {
# compute max
};
```

- Calling subroutines using references

```
&$sub_ref(23,4,54,16);
$sub_ref->(23,4,54,16);
# executes the subroutine in $sub_ref using
# supplied arguments
```

When would we ever need it?

```
my %commands = (
  "happy" => \&sing_a_song,
  "sad"   => \&call_a_friend,
  "done"  => sub { die "See ya!" },
  "mad"   => \&take_a_chill_pill
);

print "How are you? ";
chomp($string = <STDIN>);
if (exists $commands{$string}) {
  $commands{$string}->();
}
```

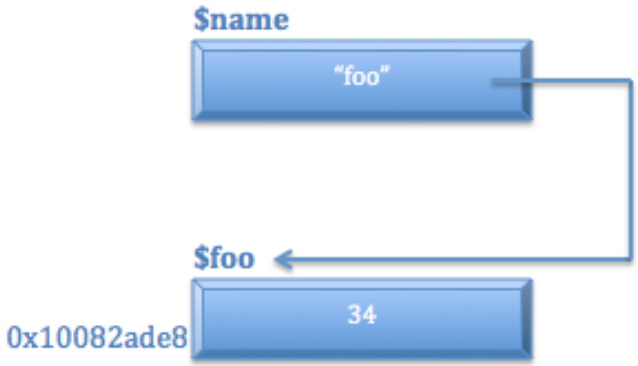
Make a copy?

```
$aref1 = [2,4,56];
$aref2 = [4,3,44];
$aref2 = $aref1; # does not copy the array
# Now, $aref2 also points to the array (2,4,56)
$aref2->[1] = 15;
print $aref1->[1]; # prints 15

# if you want to copy an array,
# create a new anonymous reference
$aref2 = [@$aref1];
# same holds for hashes
$href2 = {%{$href1}};
```

Symbolic references

- stores the name of a variable
- can be dereferenced

```
$name = "foo";
$$name = 34;
# sets $foo = 34
$name->[0] = "Jim";
# sets the element 0 of @foo
$name->{jim} = "34";
# value of "jim" in %foo
&$name;
# calls &foo
```



- could be dangerous
- `use strict 'refs';` pragma to prevent accidental use

Summary

- References
  - creating references to scalars, arrays and hashes
  - creating anonymous references
  - The arrow operator ->

```
$ref->[ ] # $ref has an array
$ref->{ } # $ref has a hash
$ref->() # $ref has a subroutine
```

- Data structures
  - arrays of arrays
  - hashes of arrays
  - hashes of hashes
  - ...
- Subroutine references
  - named and anonymous
  - calling using ->
- Symbolic references