Perl Programming

Tugba Onal-Suzek, PhD

Adapted from Dr. Jeff Solka's
BINF634 class slides

Outline

- Sorting Hashes by key and value
- Two-dimensional arrays
- Using Hashes to Pass Parameters to subroutines
- Arrays of Hashes
- Hashes of Hashes

The Comparison Operator

```
1.  $a <=> $b returns
          0 if equal
          1 if $a > $b
         -1 if $a < $b

2.   The "cmp" operator gives similar results for strings

3.   $a and $b are special global variables:
         you can NOT declare with "my" and can NOT modify.
```

Sorting Hashes by Keys

```
#!/usr/bin/perl
use strict;
use warnings;

my(%sales_amount) = ( auto=>100, kitchen=>2000, hardware=>200 );

for my $dept (sort keys %sales_amount) {
    print $dept,”: “, $sales_amount{$dept};
}

exit;
```

```
Output:
auto: 100
hardware: 200
kitchen:2000
```

Sorting Hashes by Value

```
#!/usr/bin/perl
use strict;
use warnings;

my(%sales_amount) = ( auto=>100, kitchen=>2000,
   hardware=>200 );
sub bysales { $sales_amount{$b} <=> $sales_amount{$a} }

for my $dept (sort bysales keys %sales_amount) {
   print $dept,": ", $sales_amount{$dept};
}

exit;
```

```
Output:
kitchen:2000
hardware: 200
auto: 100
```

Outline

- Sorting Hashes by key and value
- Two-dimensional arrays
- Using Hashes to Pass Parameters to subroutines
- Arrays of Hashes
- Hashes of Hashes

Two-dimensional arrays

- We can create a two-dimensional array by creating a array of lists:

```
@A = ([1, 2, 3, 4], [5, 6, 7, 8]);
```

- Access a two-dimensional array by using double brackets:

```
print "$A[1][2]\n";
7
```

- The number of rows is just the size of the array:

```
$rows = scalar @A;
```

- We can get the size of the fist row as follows:

```
# curly brackets are needed to distinguish from @$A[0]
  $cols = scalar @{$A[0]};
```

- Print out all rows and columns:

```
for ($i = 0; $i < $rows; $i++) {
 for ($j = 0; $j < $cols; $j++) {
     print "$A[$i][$j] ";
 }
 print "\n";  # newline after each row
}
```

Two-dimensional arrays

- There is no need to declare the size of an array, so arrays can be created dynamically:

```
my @A = ();
my $rows = 100;
my $cols = 100;
# create a matrix with 1's on diagonal
for ($i = 0; $i < $rows; $i++) {
 for ($j = 0; $j < $cols; $j++) {
   $A[$i][$j] = 0;
 }
 $A[$i][$i] = 1;
}
```

- Array sizes can be changed dynamically:

```
$A[0][200] = 123;   # first row now has 201 items
# but other rows are unaffected!
```

Two-dimensional arrays

- Arrays can contain any scalar values
- Two-dimensional arrays in Perl do not have to be "rectangular"
- Each row can have a different length

```
# this array has three row with lengths 3, 4
and 1
my @A = ( ["John", "Jim, "Bill"],
          [ 100, 23.5, "ATCGTTGA", \%codons ];
          [ 0 ]);
```

Example: print a 2D array

```perl
#!/usr/bin/perl
use strict;
use warnings;
# File: print_array.pl

my @A = ();

# initialize the two dimensional
   array
# with some numbers
my $rows = 6;
my $cols = 5;
for (my $i=0; $i < $rows; $i++) {
    for (my $j=0; $j < $cols;
   $j++) {
   $A[$i][$j] = $i*100 + $j*17;
    }
}

# print and quit
print_2Darray(@A);
exit;
```

```perl
# a subroutine to print out a two
# dimensional rectangular array using
# 5 digits per array element

sub print_2Darray {
    my (@a) = @_;
    my $rows = scalar @a;
    my $cols = scalar @{$a[0]};
    for (my $i=0; $i < $rows; $i++) {
      for (my $j=0; $j < $cols; $j++) {
        printf "%5d ", $a[$i][$j];
      }
      print "\n"; # newline after each row
    }
}
```

```
% print_array.pl
    0    17    34    51    68
  100   117   134   151   168
  200   217   234   251   268
  300   317   334   351   368
  400   417   434   451   468
  500   517   534   551   568
```

Example: transpose a 2D array(exchange rows and columns)

```perl
#!/usr/bin/perl
use strict;
use warnings;

# File: transpose.pl
my @A = ();

# initialize the two dimensional array
my $rows = 6;
my $cols = 5;
for (my $i=0; $i < $rows; $i++) {
    for (my $j=0; $j < $cols; $j++) {
       $A[$i][$j] = $i*100 + $j*17;
    }
}
print "A:\n";
print_2Darray(@A);

my @B = transpose(@A);

print "B:\n";
print_2Darray(@B);

exit;
```

```perl
sub transpose {
    my (@a) = @_;
    my @b = ();
    my $rows = scalar @a;
    my $cols = scalar @{$a[0]};
    for (my $i=0; $i < $rows; $i++) {
       for (my $j=0; $j < $cols; $j++) {
          $b[$j][$i] = $a[$i][$j];
       }
    }
    return @b;
}

sub print_2Darray {
    .
    .
    .
}
```

```
% transpose.pl
A:
    0     17     34     51     68
  100    117    134    151    168
  200    217    234    251    268
  300    317    334    351    368
  400    417    434    451    468
  500    517    534    551    568
B:
    0    100    200    300    400    500
   17    117    217    317    417    517
   34    134    234    334    434    534
   51    151    251    351    451    551
   68    168    268    368    468    568
```

Outline

- Sorting Hashes by key and value
- Two-dimensional arrays

- Arrays of Hashes
- Hashes of Hashes

Arrays of Hashes

```
#!/usr/bin/perl -w

#demonstrates an array of hashes;

use strict;

use warnings;

my @AoH;

my $role;

my $href;
```

```
@AoH = (
    {
       husband  => "barney",
       wife     => "betty",
       son      => "bamm bamm",
    },
    {
       husband => "george",
       wife    => "jane",
       son     => "elroy",
    },

    {
       husband => "homer",
       wife    => "marge",
       son     => "bart",
    },
  );

  print "@AoH \n";
```

Output:

```
HASH(0x22a0ac) HASH(0x229f8c) HASH(0x1846024)
```

Arrays of Hashes – Manipulating the Variables

- You can set a key/value pair of a particular hash as follows:
  - `$AoH[0]{husband} = "fred";`

- To capitalize the husband of the second array, apply a substitution:

```
$AoH[1]{husband} =~ s/(\w)/\u$1/;
```

Arrays of Hashes – How to print

```
!/usr/bin/perl -w

#demonstrates an array of hashes;

use strict;

use warnings;

my @AoH;

my $role;

my $href;

my $i;
```

```
Output:
0 is { son=bamm bamm wife=betty husband=barney }
1 is { son=elroy wife=jane husband=george }
2 is { son=bart wife=marge husband=homer }
```

Note that $# is the subscript of the last element in an array

```
@AoH = (
    {
       husband   => "barney",
       wife      => "betty",
       son       => "bamm bamm",
    },
    {
       husband  => "george",
       wife     => "jane",
       son      => "elroy",
    },

    {
       husband => "homer",
       wife    => "marge",
       son     => "bart",
    },
  );
  for $i ( 0 .. $#AoH ) {
    print "$i is { ";
    for $role ( keys %{ $AoH[$i] } ) {
         print "$role=$AoH[$i]{$role}
    ";
    }
    print "}\n";
}
```

Outline

- Sorting Hashes by key and value
- Two-dimensional arrays
- Using Hashes to Pass Parameters to subroutines
- Arrays of Hashes
- Hashes of Hashes

Hashes of Hashes

```
#!/usr/bin/perl -w

#demonstrates a hash of hashes;

use strict;

use warnings;

my $family;

my $role;

my %HoH = (
    flintstones => {
        lead      => "fred",
        pal       => "barney",
    },
    jetsons => {
        lead      => "george",
        wife      => "jane",
        "his boy" => "elroy",   # key
    quotes needed
    },
    simpsons => {
        lead      => "homer",
        wife      => "marge",
        kid       => "bart",
```

```
},
);

# print the whole thing
foreach $family ( keys %HoH ) {
    print "$family: ";
    foreach $role ( keys %{ $HoH{$family} } )
    {
         print "$role=$HoH{$family}{$role} ";
    }
    print "\n";
}
```

Output:

```
simpsons: kid=bart lead=homer wife=marge
jetsons: his boy=elroy lead=george wife=jane
flintstones: lead=fred pal=barney
```

Hashes of Hashes - Printing

```
#!/usr/bin/perl -w
#demonstrates a hash of hashes;
use strict;
use warnings;
my $family;
my $roles;
my $role;
my $person;
my %HoH = (
    flintstones => {
        lead      => "fred",
        pal       => "barney",
    },
    jetsons => {
        lead      => "george",
        wife      => "jane",
        "his boy" => "elroy",   # key
   quotes needed
    },
```

```
simpsons => {
        lead      => "homer",
        wife      => "marge",
        kid       => "bart",
    },
);
# print the whole thing, using temporaries
while ( ($family,$roles) = each %HoH ) {
    print "$family: ";
    while ( ($role,$person) = each %$roles
    ) {  # using each precludes sorting
        print "$role=$person ";
    }
    print "\n";
}
```

Output:

```
simpsons: kid=bart lead=homer
    wife=marge
jetsons: his boy=elroy lead=george
    wife=jane
flintstones: lead=fred pal=barney
```