

Perl: Arrays and Associative Arrays

a.k.a. Lists and Hashes

The HTML lecture notes have more details and examples

EIW - Perl Arrays

1

Perl Arrays

- *Ordered* list of scalar values.
- Can have any number of elements.
 - Grow automatically
 - Can have *holes* (missing elements)
- Index of first element is 0 (like C/C++)

- Perl Arrays are also called *lists*.

EIW - Perl Arrays

2

Array Literals (Constants)

- Enclose a list of scalar values inside parentheses.
 - Separate adjacent elements with a comma.

`(1, 2, 3)`

`("Joe", "Sam", "Mary", "Eric", "Joan")`

`("Apples", 4, "Oranges", 5, "Pears", 1)`

EIW - Perl Arrays

3

More Array Literals

```
( $name, $age, $weight, $height )
```

```
( $a+$b, $year+1000, "Hi" . $name )
```

 ()

The empty Array (has zero elements)

EIW - Perl Arrays

4

List Constructor

- Perl supports a notation that makes it easy to build lists that are composed of sequences of numbers:

(1..5) the same as (1,2,3,4,5)

(1.5..5.5) ← → (1.5,2.5,3.5,4.5,5.5)

(10..13,18..20) ← → (10,11,12,13,18,19,20)

EIW - Perl Arrays

5

Array Variables

- Perl array variable names start with '@'
- Array variables can use the assignment operator (just like scalars):

```
@names=("Bill","William","Billy");  
@products=("Apples","Oranges","Pears");  
@prices=(.25, .30, .35);  
@newprices = @prices;  
print @names;
```

EIW - Perl Arrays

6

Fun with arrays (fancy stuff)

```
@a=(1,2,3);  
  
@b=("hi",@a,"bye"); # @b is ("hi",1,2,3,"bye")  
  
@a=@a,4; # @a is now (1,2,3,4)  
  
($x,$y,$z)=(1,2,3); # $x is 1, $y is 2, $z is 3  
  
@x=(1..8); # @x is (1,2,3,4,5,6,7,8)  
  
($f1,$f2) = @x; # $f1 is 1, $f2 is 2
```

EIW - Perl Arrays

7

Array Element Access

- Perl uses the familiar index operator [] for array access.
- You need to put a \$ in front of an array element access (not @!) since the value of an array element is a scalar:

```
#prints first element of @names  
print $names[0];
```

EIW - Perl Arrays

8

Examples

```
@names=("Joe","Bob","Jane");  
  
print("The first name is ",$names[0]);  
  
@ages=(10,14,18);  
  
$ages[1] = $ages[1]+10;  
  
$names[2] = "Fred";
```

EIW - Perl Arrays

9

Array Growth

- Perl arrays grow automatically whenever you add an element to the end.
- If you add an element past the end of an array, the elements that have not been defined have the special Perl value *undef*.

```
$a[0]=13; $a[22]=14;
```

EIW - Perl Arrays

10

Array Size

- You can get the index of the last element in an array with the special syntax:

```
 $#arrayname
```

- It starts with '\$' since it is a scalar value!

```
@x=(18,22,45);  
print $#x;
```

- This would print 2, since the index of the last element is 2 (\$x[2] is 45).

EIW - Perl Arrays

11

Array Length

- You can find out how many elements are in a Perl array by assigning an entire array to a scalar variable.

```
$len = @foo;
```

- `$len` is now the number of elements in the array `@foo`.

EIW - Perl Arrays

12

Array Example

```
#define an array
@nums=(1,2,3,5,8,13,21);

#find out the last index of @nums
$lastindex=$#nums;

#find out the number of elements
$arraysize = @nums;

print "last index is $lastindex\n";
print "number of elements is $arraysize\n";
```

EIW - Perl Arrays

13

Array Operators

push: add element to end of an array
pop: remove last element
shift: remove first element
unshift: add element at the beginning
reverse: rearrange in reverse order
sort: order elements by element value
chop: chop every element in an array

EIW - Perl Arrays

14

Array Operator usage

- **push** & **pop** can implement a stack.
- **push** & **shift** can implement a queue.
- **reverse** and **sort** do not change an array, they create a new array!

```
sort @names ← sorts and throws away the result!
@names = sort @names; ← saves the result
```

EIW - Perl Arrays

15

<STDIN> as an Array

- You can assign <STDIN> to an array!
- Perl reads *all remaining lines of input*, each line becomes an element in the array.
- *all remaining* usually means until End-of-File is found.
 - everything is in the array – you shouldn't expect to be able to read anything else from STDIN!

EIW - Perl Arrays

16

Example

```
print "Enter your name, age and weight,
      each on a separate line\n";
@lines = <STDIN>;
chop(@lines);
($name,$age,$weight) = @lines;

print "Your name is $name\n";
print "Your age is $age\n";
print "Your weight is $weight\n";
```

EIW - Perl Arrays

17

Variable Interpolation of Arrays

- Array name is replaced by a list of the elements with spaces between each:

```
@problems=("the","lack","of","parking");
print "the problem with RPI is @problems\n";

# will print the problem with RPI is the lack
of parking
```

EIW - Perl Arrays

18

Associative Arrays (hashes)

- Collection of scalar values that can be accessed by an index.
 - there is no order to the collection.
 - the index can be anything, not just integers starting at 0.
 - typically the index is a string
 - we usually call the index *the key*

EIW - Perl Arrays

19

Assoc. Array Variables

- Assoc. Array variable names start with ‘%’
- The index operator is { }
- Accessing an assoc. array element is a *scalar* operation (so an individual element starts with ‘\$’)

EIW - Perl Arrays

20

Assoc. Array Examples

```
      key           value
      ↓             ↓
$age{"John Smith"}=32;
$age{"Mary Jones"} = 45;
$age{"RPI"}=180;

print $age{"John Smith"}; # prints 45

$x = "RPI";
print $age{$x}; # prints 180
```

EIW - Perl Arrays

21

Another Example

```
$text{0}="zero";
$text{1}="one";
  # and so on
$text{9}="nine";

print "Enter a number\n";
$v=<STDIN>;
print "The name of " . $v . " is " .
      $text{$v} . "\n";
```

EIW - Perl Arrays

22

Another Example

```
$val{"zero"}=0;
$val{"one"}=1;
  # and so on
$val{"nine"}=9;

print "Enter the name of a number\n";
$n=<STDIN>;
print "The value of $n is $val{$n}\n";
```

EIW - Perl Arrays

23

Assoc. Array Operators

keys: returns a *list* of the keys (indices)
`@thekeys = keys %age;`

values: returns a *list* of the values
`@thevals = values %age;`

delete: remove an element from an assoc.
array.
`delete $age{"John Smith"};`

EIW - Perl Arrays

24

Variable Interpolation and Associative Arrays

- No variable interpolation for an entire associative array.
- You can do variable interpolation on an individual assoc. array element:

```
print "Age: $age{John Smith}\n";
```

EIW - Perl Arrays

25

Assoc. Array Literals

- You can assign a list to a hash variable, every other element will become an index:

```
%names = (0, "zero", 1, "one", 2, "two");  
print("The name of 1 is $names{1}\n");
```

```
%vals = ("zero", 0, "one", 1, "two", 2);  
print "The value of one is $vals{one}\n";
```

EIW - Perl Arrays

26

Alternate Syntax

- You can also use the following syntax:

```
%names = (0=>"zero", 1=>"one", 2=>"two");  
print("The name of 1 is $names{1}\n");
```

quotes are optional here!

```
%vals = ("zero"=>0, "one"=>1, "two"=>2);  
print "The value of one is $vals{one}\n";
```

EIW - Perl Arrays

27

Exercise

- Write some Perl code that will create an HTML table, where the contents of the cells come from a Perl Array.

EIW - Perl Arrays

28

One Solution

```
@vals=("one","two","three");  
# assume the array @vals has the values  
print "<TABLE><TR>\n";  
for ($i=0;$i<=$#vals;$i++) {  
    print "<TD>$vals[$i]</TD>\n";  
}  
print "</TR></TABLE>";
```

produces this



```
<TABLE><TR>  
<TD>one</TD>  
<TD>two</TD>  
<TD>three</TD>  
</TR></TABLE>
```

EIW - Perl Arrays

29

Another Exercise

- Print a paragraph tag where the attribute names and values are specified in a Perl hash (associative array).

EIW - Perl Arrays

30

One Solution

```
%attrs = (  
  "ID" => "joe",  
  "style" => "color:blue;",  
  "onClick" => "check();"  
);  
  
printf("<P ");  
foreach $attr (keys %attrs) {  
  print "$attr=\"${attrs{$attr}}\" ";  
}  
print ">";
```

produces this

```
<P style="color:blue;" onClick="check();" ID="joe" >
```

EIW - Perl Arrays

31
