Arrays and Hashes in Perl
Kevin Jones

Introduction

This document describes the use of data structures in Perl, and their higher abstractions. I start off by describing the array and hash structures. It then the document gets more complex by investigating array of arrays, array of hash, hash of hash, and hash of array.

Arrays

Unlike scalar variables in Perl, arrays are assigned with the "@" symbol. For example,

```
@my_array = ("one", "two", "three");
```

creates an array, whereas this

```
$my_array = ("one", "two", "three");
```

Would set the value to my $my_array to "three". The comma value indicates a list context. You can also declare an array by using the *qw* syntax as follows:

```
@my_array = qw(one two three);
```

So now that we have learned how to create an array, you need to know how to determine the size of the array. One way to do this is by evaluating your array in a scalar context. Examples:

```
scalar(@my_array);
@my_array + 0;
$#my_array + 1;
```

You may extend or shrink an array using this last method, for example.

```
# clearing arrays
@my_array = ();
$#my_array = -1;

# extending arrays
$#my_array = 5 # sets index 0 to 4
```

When you truncate an array, you do not free the memory. To free the memory you have to use:

```
undef(@my_array);
```

To access the elements in an array in a scalar context you use brackets.

```
$my_array[1] = "one";
```

Hashes

Hashes are similar to arrays in many ways. One difference between them is that an array is indexed by integer values; whereas the hash uses key strings to look up values. Also, hashes are not ordered, unlike arrays. In Perl, hashes are defined with the "%" symbol, and they are created as follows. All three examples have the same outcome.

```
# Example 1
%hash_grades = ('John', 95, 'Ben', 88, 'Alice', 98);

# Example 2
%hash_grades        = ();
%hash_grades{John}  = 95;
%hash_grades{Ben}   = 88;
%hash_grades{Alice} = 98;

# Example 3
%hash_grades = (John  => 95,
                Ben   => 88,
                Alice => 98);
```

If you evaluate a hash in a scalar context, it returns true only if it contains any values. To find the total number of keys in a hash, you use the keys function.

```
 (scalar(keys(%my_hash))).
```

Arrays of Arrays

Arrays of arrays or multi-dimensional arrays are created in the following manner.

```
# The following assigns a list of array references to an array.
@AoA = ( [ "red", "blue" ],
         [ "one", "two", "three" ],
         [ "Dallas", "Houston", "College Station" ],
);

# and is used in the following way
print $AoA[2][1];   # Houston

# The following creates a reference to an array of array references.
$ref_to_AoA = [
    [ "red", "blue", ],
    [ "one", "two", "three", ],
    [ "Dallas", "Houston", "College Station", ],
];

print $ref_to_AoA->[1][2];   # three
```

There is an implied -> between every pair of brackets, but there is not a -> implied before the first pair of brackets. When you index an array, you can count backward using a

negative integer. In order to extend the rows of an array you can us the *push* function to add an array reference to an array. To add a new column to an array you can just make an assignment. To access an element in the array you need to use multiple brackets according to how many dimensions of your array you have.

Arrays of Hashes

The following creates an array of hashes and shows how to populate it.

```
# Create it
@AoH = ({name => “joe”,
         grade => 98, },);

# Add data to it
push @AoH {name => "mike", grade => 87};

# Add Keys
$AoH[0]{id} = “0392302932”;
```

Hash of Hash

Hashes of hashes are very flexible; however since hashes are hashed, they cannot be sorted. The following demonstrates how to create and populate a hashes of hashes structure.

The following creates an hash of hashes and shows how to populate it.

```
# Create it
@HoH = (final = > {name => “joe”, grade => 98, },);

# Add Keys
@HoH{midterm} = {name => "mike", grade => 87};

# Add data
$HoH{final}{name} =  “linda”
```

Hash of Array

```
# Create it
@HoA = (joe => [91, 93, 84],
        mike => [95, 84, 94],);

# Add an array
@HoH{alice} = [93, 93, 84];
```

Conclusion

When working in Perl you need to keep in mind what context you are working in. It is difficult to program in Perl if one cannot understand the difference between the list context and scalar. The creation of memory is automatic in Perl which allows the

programmer to concentrate on the functionality of the program rather than the memory aspects of programming.

References

- Christiansen, Tom; Orwant, Jon; Wall, Larry; "Programming Perl, 3rd Edition", O'Reilly
- http://faculty.cs.tamu.edu/yurttas/PL/SL/perl/