IT441

Network Services Administration

Hashes

The Transliteration Operator

- This string operator resembles the substitute operator from a *regex* – **`s///`** – but it functions very differently
- **`tr/old/new/`** will not replace occurrences of the string ***old*** with the string ***new***
- It will replace
  - any occurrence of the letter **`o`** with the letter **`n`**
  - any occurrence of the letter **`l`** with the letter **`e`**
  - any occurrence of the letter **`d`** with the letter **`w`**

The Transliteration Operator

- How can we use the transliteration operator to change every occurrence of a ***comma*** into a ***period***?
- How can we use the transliteration operator to change every occurrence of the letter `a` to the number `1`?
- How can we use the transliteration operator to change the ***case*** of all letters in a string?
- Write a small program to:
  - Input the file `string.txt` into your program
  - Change all occurrences of small letters into capital letters
  - Print the resulting string to the screen

Hashes

- We have already studied two ways to store and organize data in a Perl program:
  - Scalars
  - Arrays
- Each has its own strengths and weaknesses
- In Perl (and many other programming environments) there is a third way to store data. It is referred to as a *HASH*
- (It is comparable to a *dictionary* in Python.)
- Hashes are very useful and very powerful!

Data Structures

- Remember that a scalar variable always starts with a **$** , for example **`$scalar`**
- Remember that an array structure always starts with a **@** , for example **`@array`**
- Well a hash always starts with a **%** , for example **`%hash`**

What is a HASH?

- A ***hash*** is a data structure that consists of pairs of datum, one called the key and one called the value.
- Some people call a hash an *associative array*.
- A hash is stored in no particular order.
- In a hash...
  - the *keys* must be ***unique***,
  - but the *values* have no such restrictions.

What Can We Use a Hash For?

- A classic example of a use for a hash in Systems Administration is the MAC⇔IP pairing
- To create this type of hash we would enter:

```
my %ipMac = (
 '192.168.124.1', '2a:09:4e:3c:31:42'
 '192.168.124.2', '0e:88:4e:2a:56:07'
 '192.168.124.3', '1a:32:6f:5c:6b:1a'
 );
```

- This hash uses the IP as the *key* and the MAC as the *value*.

What Can We Use a Hash For?

- We could also enter the previous hash as

```
my %ipMac = (
 192.168.124.1 => '2a:09:4e:3c:31:42'
 192.168.124.2 => '0e:88:4e:2a:56:07'
 192.168.124.3 => '1a:32:6f:5c:6b:1a'
 );
```

- We call the symbol *=>* the *quoting comma* because it acts as a comma and quotes the string to the left of it.

What Can We Use a Hash For?

- Another possible use for a hash is using student numbers as the key and student names as the value.
  - Why could we not reverse the key, value pairs in this case?

- Can you think of some other examples where a hash would be a good choice for a data structure?

Another Way to Define the Hash

- Since hashes and arrays have a lot in common we can change back and forth between them.
- Given the following array:

```
@array = qw( Thao Saigon Hoang Boston Kevin
Dorchester Gary Houston Susan Manchester Allison
Chicago)
```

- We can create a hash directly by an assignment statement:

```
%where = @array
```

- What are the pairs in this hash?

Another Way to Define the Hash

- This way of defining an hash is equivalent to the following:

```
%where = (
      Thao => 'Saigon'
      Hoang => 'Boston'
      Kevin => 'Dorchester'
      Gary => 'Houston'
      Susan => 'Manchester'
      Allison => 'Chicago'
      );
```

Working with Hash Values

- To look up a value in a hash we use something similar to the index notation for an array. However there are two (2) differences:
  - Instead of locating the value by number, we locate it by the key
  - Instead of using square brackets `[ ]` we use curly braces `{ }`

- Here is an example using the hash %where:

```
print "Hoang lives in $where{Hoang}";
```

Modifying the Hash

- Adding a new element to the hash is very simple. All that is needed is to type an assignment statement:

```
$where{Al} = "Quincy";
```

- Changing an element in the hash is just as simple:

```
$where{Kevin} = "Quincy";
```

  - Remember, Kevin used to live in Dorchester but he now lives in Quincy

- Or we can remove an element from the hash using the delete function:

```
delete $where{Gary} ;
```

  - Will remove Gary and Houston from the hash

Hash Functions

- There are a few functions that operate on hashes.
  - **`keys(%hash)`** returns a list of *all keys* in **`%hash`**
  - **`values(%hash)`** returns a list of *all values* in **`%hash`**
  - **`each(%hash)`** returns *each* key/value *pair* in **`%hash`**
  - **`delete $hash{key}`** *deletes* the key/value pair in **`%hash`**
  - **`exists $hash{key}`** returns ***true*** if a entry with that key *exists* in **`%hash`**