**CSC105, Introduction to Computer Science**
Lecture: Perl Hash Data Structure

A **hash** is a set of *key*/*value* pairs. The pairs denote a mapping from the first item (usually called the *key*) to the second (the *value*).

You can think of a hash as an index in which some term (a string) is used to tag or identify some value. The value itself can be any scalar or complex data structure. For example, suppose we wish to make a lookup table for a set of associated values that list the number of congressmen elected in each state.

| | |
|---|---|
| **Alabama** | **7** |
| **Alaska** | 1 |
| **Arizona** | 9 |
| **Arkansas** | 4 |
| **California** | 53 |
| **Colorado** | 7 |
| **Connecticut** | 5 |
| **Delaware** | 1 |
| **…** | … |
| **Wyoming** | 1 |
| *key* | *value* |

Figure 1: Perl hash data structure mapping states and their number of elected representatives.

Unlike arrays or lists that must be searched linearly by position, supplying a unique key magically fetches its associated value.

Programming hashes in Perl resembles the techniques that you used with lists. In general, you must

- designate the hash using a special variable whose prefix is the % (percent) sign,

  here are some examples: **`%map`** **`%index`** **`%representatives`**
- define the hash as a set of key/value pairs,
- process the items in the hash using repetition.

Creates a simple hash associating food with its characteristic color and prints the results.

```
#! usr/bin/perl
# perform process with keys and values
%food_color = (
  Apple => "red",
  Banana => "yellow",
  Lemon => "yellow",
  Carrot => "orange"
);
# loop through hash
foreach $food (keys %food_color){
  $color = $food_color{$food};
  print "$food is $color\n";
}
```

output:

Banana is yellow
Apple is red
Carrot is orange
Lemon is yellow

It is also easy to insert new items into the hash. However—perhaps, surprisingly—you have no control over where the item will be placed in the structure.

```
#! usr/bin/perl
# insert into a hash
%food_color = (
  Apple => "red",
  Banana => "yellow",
  Lemon => "yellow",
  Carrot => "orange"
);
print "Known foods:\n";
foreach $food (keys %food_color){
  print "$food\n";
}
# insert new item
$food_color{Spinach} = "green";
print "New list:\n";
foreach $food (keys %food_color){
  print "$food\n";
}
```

which produces the following output:

```
Known foods:
Banana
Apple
Carrot
Lemon
New list:
Banana
Apple
Carrot
Spinach
Lemon
```

As you can see the key “Spinach” was added, but its placement in the hash is unpredictable. This is due to the internals of its implementation.

It is possible to sort the hash in a variety of ways. First, we sort it using the key.

```
#! usr/bin/perl
# print sorted hash
%food_color = (
  Apple => "red",
  Banana => "yellow",
  Lemon => "yellow",
  Carrot => "orange"
);
# loop through hash
foreach $food (sort keys %food_color){
  $color = $food_color{$food};
  print "$food is $color\n";
}
```

output:

```
Apple is red
Banana is yellow
Carrot is orange
Lemon is yellow
```

Next, we sort it by value.

```
#! usr/bin/perl
# print sorted hash by values
%food_color = (
  Apple => "red",
  Banana => "yellow",
  Lettuce => "green",
  Lemon => "yellow",
  Carrot => "orange"
);
# loop through hash
foreach $food (sort { $food_color{$a} cmp
$food_color{$b} }
    keys %food_color){
  print "$food is $food_color{$food}\n";
}
```

output:

Lettuce is green
Carrot is orange
Apple is red
Banana is yellow
Lemon is yellow

Last, one of the most common applications for the hash structure is to count the number of instances found in a collection. For example, suppose that we are interested in finding out how many times each word appears in a list.

```
#! usr/bin/perl
# count how many times an element occurs in a list
@list = qw(apple banana pear apple orange lemon lime
  lemon banana apple);

%count = ( ); # empty hash
foreach $element (@list) {
  $count{$element}++; # add 1 to the current count
}
foreach $x (sort keys %count){
  print "$x = $count{$x}\n";
}
```

will produce this result:

```
apple = 3
banana = 2
lemon = 2
lime = 1
orange = 1
pear = 1
```

*NOTE*: The results are also sorted before printing.

As you can see, each value in the list is added to the hash as a key. The value is stored as an integer. When the key is first inserted, the value is incremented from 0 to 1. After that, each additional occurrence causes the value to increase by 1.