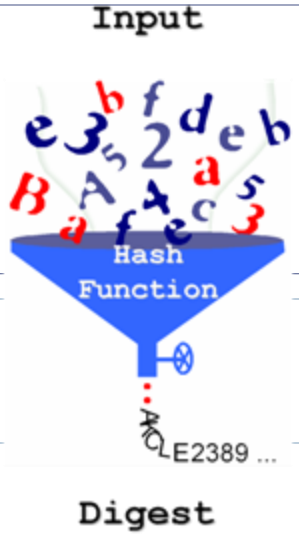
# COMP 2021

## Unix and Script Programming




## Perl Hashes

# What is a Hash?

- A *hash* (or *associative array*) is a set of key/value pairs

- A hash index is called a *key* (keys must be unique)
- Key can be any scalar value (not just small non-negative integers)
- The elements of a hash have no fixed order
- The keys are used to lookup the values.

| | | | |
|---|---|---|---|
| **keys:** | **"pi"** | **1A** | **"0 C"** |
| **values:** | **3.1415** | **2046.1** | **"32 F"** |

# Hash Variables

- Hash variable names begin with the percent sign (`%`) followed by the usual variable name.
- There is no relationship between `$data`, `@data`, and `%data`, Perl considers them to be separate variables.
- Each element of a hash is a separate scalar variable, accessed by the key.
- Elements of the hash `%data`, are referenced with `$data{$key}`, where `$key` is any scalar expression.

# Create Hashes

- Method 1: assign a value to a named key on a one-by-one basis

```
%data;
$data{"John Paul"} = 45;
$data{"Lisa"} = 30;
$data{"Kumar"} = 40;
```

- Method 2: Use list

```
%data = ("John Paul", 45, "Lisa", 30, "Kumar", 40);
```

- For clarity, you can use => as a alias for , to indicate the key/value pairs

```
%data = ("John Paul" => 45, "Lisa" => 30, "Kumar" => 40);
```

# Access Hash Elements

- Prefix the variable with `$`, and then append the element key with `{ }`

```
%data = ("John Paul" => 45, "Lisa" => 30, "Kumar" => 40);
print "$data{"John Paul"}\n";
print "$data{"Lisa"}\n";
print "$data{"Kumar"}\n";
```

- You can get a list of all keys or values by using `keys()` and `values()` function correspondingly

```
@names = keys %data; # when use keys(), () is optional
@ages = values %data;
print "$names[0] is $ages[0] years old.\n";
```

# keys

- The `keys` function is often used in `foreach` loops to print or access the elements one-by-one:

```
$cat sorthash.pl
#!/usr/local/bin/perl5 -w

my %planets = (
Mercury => 0.4,
Venus => 0.7,
Earth => 1,
Mars => 1.5,
Ceres => 2.77,
Jupiter => 5.2,
Saturn => 9.5,
Uranus => 19.6,
Neptune => 30,
Pluto => 39,
Charon => 39,
);

foreach my $name (keys %planets) {
printf "%-8s %s\n", $name, $planets{$name};
}
```

**Output:**

```
Mercury  0.4
Ceres    2.77
Uranus   19.6
Earth    1
Pluto    39
Saturn   9.5
Venus    0.7
Mars     1.5
Neptune  30
Jupiter  5.2
Charon   39
```

# More on `keys/values`

- If there are no elements in the hash, then `keys/values` returns an empty list.
- `Keys` returns an array of keys from the hash
- `Values` returns an array of values from the hash
- The order of the two arrays will match
- But you can't tell the exact order

```
%hash = ( a=>1, b=>2, c=>3);
@v = values %hash; # e.g. ( 2, 3, 1 )
@k = keys %hash; # e.g. ('b', 'c', 'a')
```

- The size of a hash (e.g. the number of key-value pairs) can be obtained by `scalar keys %hash`

# Hash Slices

- We can pick out a slice of elements from hash
- Get a single element from a hash with key `$key`

```
$value = $hash{$key}
```

- Get a list of elements from the same hash, referred to by the keys in `@keys`

```
@values = @hash{@keys}

%hash = ("one"=>1, "two"=>2, "three"=>3,
"four"=>4);
($two, $four) = @hash("two", "four");
```

# Add & Remove Hash Elements

- Adding a new key/value pair uses simple assignment

```
$hash{newkey} = newvalue;
```

- `delete` function is needed to remove hash elements
  - Removes key and its associated value

```
delete $hash{key};
```

  - Gets rid of a slice of key/value pairs at once;

```
delete @hash {"first","second","third"};
```

- To empty out a hash

```
delete @hash{keys %hash}; # Hash slice
%hash = (); # Most efficient method
```

# Add & Remove Example

```
$ cat addremovehash.pl
#!/usr/local/bin/perl5 -w

%data = ("John Paul" => 45, "Lisa" => 30, "Kumar" => 40);
$size = scalar keys %data;
print "1 - Hash size:  is $size\n";

# adding an element to the hash;
$data{"Ali"} = 55;
@keys = keys %data;
$size = @keys;
print "2 - Hash size:  is $size\n";

# delete the same element from the hash;
delete $data{"Ali"};
$size = scalar keys %data;
print "3 - Hash size:  is $size\n";
```

Output:

```
1 - Hash size:  is 3
2 - Hash size:  is 4
3 - Hash size:  is 3
```

# Print Hashes

- You cannot print the entire hash like you can arrays:
- You can use a `foreach` loop to print hashes
- You can also use `Dumper` to print hashes

```
$cat printhash.pl
#!/usr/local/bin/perl5 -w
use Data::Dumper;
my %hash = (yat=>1, yee=>2, saam=>3);

print "Cantonese hash: %hash\n";

foreach (keys %hash){
        print "$_: $hash{$_}\n";
};

print Dumper(\%hash);
```

Output:

```
Cantonese hash: %hash

saam: 3
yee: 2
yat: 1

$VAR1 = {
          'saam' => 3,
          'yee' => 2,
          'yat' => 1
        };
```

# Hash `sort`

- `sort()` function can be used to sort keys/values in hash
  - **Default sort is based on ASCII table !!!**

```
foreach my $name (sort keys %planets) {
      printf "%-8s %s\n", $name, $planets{$name};
}

foreach my $distance (sort values %planets) {
      print "$distance\n";
}
```

- We may want to override ASCII sorting sometimes
  - Sorting in alphabetical or numerical order
- Refer to `sorthash.pl` for more details and comparison

# Hash `reverse`

- You can construct a hash with keys and values swapped using the `reverse` function:

```
my %original;
%inverse = reverse %original;
```

- Note with `reverse`, duplicate values will be overwritten
- Has to unwind one hash and build a whole new one

# Example: Create Hash from File

```
$cat hashfromfile.pl
#!/usr/local/bin/perl5 -w

open (FILE, "$ARGV[0]") or die "$!\n";
my %hash;
while (<FILE>) {
        chomp;
        @row = split ("\t");
        if (scalar(@row) ==  2){
                $key = $row[0];
                $value = $row[1];
                $hash{$key} = $value;
        }
}
foreach (keys %hash) {
        print "$_ has value $hash{$_}\n";
}
```

# Example: Getting Keys from Values

```
$cat keyswithhashvalues.pl
#!/usr/local/bin/perl5 -w
use Data::Dumper;

my %hash = (
   "apple"  => "red",
   "banana" => "yellow",
   "orange" => "orange",
   "lemon"  => "yellow",
);
```

```
print "Hash with duplicate values:\n";
print Dumper(\%hash);

my $value = 'yellow';
print "Search for fruit with yellow color:\n\n";

print "Method 1: reverse\n";
my %revhash = reverse %hash;
print "$revhash{$value}\n\n";

print "Method 2: grep\n";
my @matching_keys = grep { $hash{$_} eq $value } keys %hash;
print("$_\n")
   foreach @matching_keys;
```