Brandon Green
CPSC 332-300
7-9-04
Project 01

Using Perl Arrays and Hashes

Using arrays and hashes in perl is not difficult, but in order to combine arrays and hashed to hold data you have to know a little bit about how perl holds references to the different data types. The most basic examples are to just use arrays and hashes by themselves:

```
sub array_demo()
{
  my $size = 8;
  my @array;
  my $i;

  for($i=0;$i<=$size;$i++)
  {
    $array[$i]=$i*$i;
  }

  print "Squares from 0 to $size: ";
  for($i=0;$i<=$size;$i++)
  {
    print " $array[$i]";
  }

# could of initialized as @array=(0,1,4,9,16 ...
}
```

Simple variables in perl are prefixed with $, but arrays use the prefix @ to specify that they are arrays. In addition to not needing to predefine the size of the array(Perl will expand it as necessary), you can also perform push and pop operation on the array, and you can find the size of the array by using the scalar function or simply $#array.

```
sub hash_demo()
{
  my $size = 8;
  my %hash;
  my $i;

  for($i=0;$i<=$size;$i++)
  {
    $hash{"$i squared"} = $i*$i;
  }

  print "Squares from 0 to $size: ";
  foreach $i (keys(%hash))
  {
    print " $i=$hash{$i},";
  }

# could of initialized as %hash=( "0 squared" => 0, "1 squared" => 1, ...
}
```

Hashes can be very useful because they allow the association of a string to a piece of data, this relaxes the requirement of an array where everything must be indexed by a preset number determined by its position in the array. In perl a hash is designated with a %, and the data in the hash can be initialized when it is declared or additional elements can be added to the hash.

```
sub array_of_array_demo()
{
  my $size=8;
  my @array;
  my @tarray;
  my $i;
  my $j;

  for($i=0;$i<=$size;$i++)
  {
    for($j=0;$j<=$size;$j++)
    {
      $tarray[$j]=$i*$j;
      if ($i==0)
      {
        $tarray[$j]=$j;
      }
      if ($j==0)
      {
        $tarray[$j]=$i;
      }
    }
    push @array, [ @tarray ];
  }

  print "Multiplication table up to $size\n";
  for($i=0;$i<=$size;$i++)
  {
    for($j=0;$j<=$size;$j++)
    {
      print "\t$array[$i][$j]";
    }
    print "\n";
  }

# could of initialized as @array= ([1,2,3],[4,5,6], ... etc);
}
```

When creating an array of arrays dynamically, you just have to insert into the array references to the other array, since perl automatically handles assigning of different data types, you don't have to declare anywhere that you want to create an array of arrays, you just simply add some arrays into an array.

```
sub array_of_hash_demo()
{
  my @array_hash = (
                {"file" => "hw01",  "size"  => "120 B",  "creator"  => "alpha"},
                {"file" => "hw02",  "size"  => "340 B",  "creator"  => "alpha"},
                {"file" => "hw03",  "size"  => "750 B",  "creator"  => "alpha"},
                {"file" => "hw11",  "size"  => "182 B",  "creator"  => "beta"});

  my $array_size = scalar( @array_hash );
  my $i;
```

```
  for($i=0; $i<$array_size; $i++)
  {
    print "File - $array_hash[$i]{'file'} Size - $array_hash[$i]{'size'} Creator - $array_hash[$i]{'creator'}\n";
  }
}
```

An array of hashes can be useful when you want to store data about something in a hash, and need to store multiple occurances of the data. In the example an array of hashes that hold information about files is created, and then the data is output by going through all of the hashes in the array.

```
sub hash_of_hash_demo()
{
  my %hash_hash = (
          "Texas"  => { "Houston" => 4986399, "Dallas" => 5484061, "Austin" => 1349291, "San Antonio" =>
1786620 },
          "Florida"  => { "Tallahassee" => 327869, "Miami" => 5232107, "Orlando" => 1752192 },
          "California"  => { "Los Angeles" => 12745084, "San Diego" => 2906660, "San Francisco" => 4179500 });

  my $hash_size = scalar(keys(%hash_hash));
  my $i;
  my $j;

#  print $hash_hash{"Texas"}->{"Houston"};
  foreach $i (keys(%hash_hash))
  {
    foreach $j ( keys(%{$hash_hash{$i}} ))
    {
      print "$i \t $j \t $hash_hash{$i}{$j}\n";
    }
  }
}
```

Using a hash of a hash you can store a hierarchy of information in a very easy to read format. In the example a list of city populations is organized by states into one hash. When accessing the first level hash $hash_hash{$key} you get a reference to another hash, so when you access it with the key from the second hash, you get the data element stored there, which in this case is the population of that city.

```
sub hash_of_array_demo()
{
  my %hash_array = (
          "Factors of 4"  => [1,2,4],
          "Factors of 6"  => [1,2,3,6],
          "Factors of 8"  => [1,2,4,8],
          "Factors of 9"  => [1,3,9],
          "Factors of 10"  => [1,2,5,10],
          "Factors of 12"  => [1,2,3,4,6,12],
          "Factors of 14"  => [1,2,7,14],
          "Factors of 15"  => [1,3,5,15],
          "Factors of 16"  => [1,2,4,8,16]);

  my $array_size;
  my $i;
  my $j;

  foreach $i (keys(%hash_array))
  {
```

```
    print "$i - ";

    #scalar(@{$hash_array{$i}}); also works
    $array_size=$#{$hash_array{$i}};

    for($j=0; $j<$array_size; $j++)
    {
      print "$hash_array{$i}[$j] ";
    }
    print "\n";
  }
}
```

The hash of arrays is simply a hash that instead of storing a piece of data like a string holds an array. Accessing elements in the hash of arrays is as simple as above. One thing to note is that when getting the size of the array stored, you must do $#{$hash_array {$key}} because $hash_array{$key} returns a reference to an array, so the extra braces are needed to tell it to access what is referenced to.