LAB 2 – Perl II

Installation

OBS: On the course computers these programs are already installed!

ActivePerl: http://www.activestate.com/activeperl

Open Perl IDE: http://open-perl-ide.sourceforge.net/

Task 1 - Basics

a) Hashes

Hashes let us store key-value pairs:

```
my %cities;
foreach ("Oslo","Trondheim", "Bergen") {
   $cities{$_} = "Norway";
}
foreach ("Stockholm","Malmö", "Kiruna") {
   $cities{$_} = "Sweden";
}

print "Oslo is in $cities{Oslo} and Stockholm in $cities{Stockholm} you
ignorant ----!\n";
```

Notice that while elements in arrays are accessed by an integer index (e.g. `$cities[2]`), values in hashes are accessed by a key of any type (e.g. `$cities{Stockholm}`). All keys can be retrieved using `keys`:

```
my @cities = keys %cities;
print "@cities\n";
```

The existence of a key is checked using `exists`:

```
my @check = ("Umeå","Malmö");
foreach (@check) {
  if (exists $cities{$_}) {
    print "$_ is stored\n";
  } else {
    print "$_ is not stored\n";
  }
}
```

b) Data structures

Scalers can be references that point to any data type and can thus be used to build complex data structures. In practice, we typically don't think of these references. Instead, we think of hashes of hashes, hashes of arrays, arrays of arrays, etc. This is an example of hashes of hashes:

```
my %cities;
foreach ("Oslo","Trondheim","Bergen") {
   $cities{"Norway"}{$_} = 1;
}
foreach ("Stockholm","Malmö","Kiruna") {
   $cities{"Sweden"}{$_} = 1;
}
```

Notice how countries and cities are stored as pairs of keys, while the values are unimportant (i.e. 1 for all pairs of keys). `exists` can be used to check also the existence of pairs :

```
print "Is Kiruna in Norway:";
if (exists $cities{Norway}{Kiruna}) {
  print " YES!\n";
} else {
  print " NO!\n";
}

print "Is Oslo in Norway:";
if (exists $cities{Norway}{Oslo}) {
  print " YES!\n";
} else {
  print " NO!\n";
}
```

Checking for exists of two keys can be dangerous: if the first key does not exist, it will be created! Try `exists $cities{Finland}{Stockholm}` followed by `exists $cities{Finland}`. How would you solve this problem?

c) Subroutines

Subroutines are great ways of making your program shorter, easier to read and maintain … and much more:

```
my %cities;
foreach ("Oslo","Trondheim","Bergen") {
   $cities{"Norway"}{$_} = 1;
}
foreach ("Stockholm","Malmö","Kiruna") {
   $cities{"Sweden"}{$_} = 1;
}

print "Is Kiruna in Norway:";
isin(\%cities,"Kiruna","Norway");

print "Is Oslo in Norway:";
isin(\%cities,"Oslo","Norway");

sub isin {

  my $hash = $_[0];
  my $city = $_[1];
  my $country = $_[2];

  if (exists $hash->{$country}{$city}) {
    print " YES\n";
  } else {
    print " NO\n";
  }
}
```

Task 2 - Hashes

a) Write a program that remove**s** duplicates from a list using hashes (this assignment was also given in Lab1).

b) Write a program that finds the intersection of two sets using hashes (an intersection of two sets is the subset of elements that is in both sets/lists).

Task 3 - Nested data structures

a) Write a program that reads the gene expression values from *microarray.txt* into a table. If the user input a single index (e.g. 3), then output the row corresponding to that index. If the user inputs two

indices (e.g. 3 2), then output the value corresponding to that entry in the table. Output error messages if the indices are not valid.

b) The file *network.txt* lists regulatory interactions. The first column contains transcription factor, the second column contains the type of interaction (positive or negative **regulation**) and the third column contains the gene module regulated by the transcription factor.

Write a program that reads the content of the file and store it as hashes of hashes. If the user inputs a transcription factor, then output all modules it regulates. If the user inputs a module, output the transcription factor(s) that regulate it. Output error messages if the names do not exist.

Hint: Store the data as two hashes of hash. One where transcription factors is the first key and modules are the second key, and one where modules are the first key and transcription factors the second.

Task 4 - Subroutines

a) Write a subroutine that takes three arguments: a table, index i, index j and a value. The subroutine should change the element (i,j) to the given value in the table. Make two versions: One using call-by-value and one using call-by-reference. Use the code and table in Task3a to test the subroutines.

b) Repeat Task 4a but move the two subroutines to a separate module. Again use the code and table in Task3a to test the module.

To get the lab approved, send your code to: jenny.onskog (at)plantphys.umu.se