LAB 1 – Perl I

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 - Your first Perl programs

a) Hello world

In Open Perl IDE, type the following program:

```
use strict;
use warnings;

print "Hello world!\n";
```

`strict` and `warnings` are packages that help finding bugs in your programs. They should always be included. The `print` command prints to screen whatever is inside the quotation marks. `\n` prints a new line.

Save the program to file, e.g. `Task1a.pl` (`.pl` is the common file extension for Perl programs and Open Perl IDE requires this extension). Run the program from the menu: **Run->Run**.

b) Variables

Type the following program:

```
my $x = 5;
print "x = $x\n";
```

Remember to add `strict` and `warnings`. We will skip them in the following examples to save space.

`$x` is a variable, called a scalar, that can hold any value (string, number, etc). `print` will not print "`$x`", but rather the value of the scalar `$x`. Try running the program.

Now add these two lines to the program:

```
$x = "five";
print "x = $x\n";
```

Scalars can be reassigned to hold other values, including values of other types. Try it.

Scalars can also be added, subtracted, etc. Add these lines and see what happens:

```
$x = 5;
my $y = 8;
my $z = $x + $y;
print "$x + $y = $z\n";
```

What happens if you change `$y` to a string (e.g. “eight”)? Note: Open Perl IDE prints normal output to **Console**, but prints error messages to **Error Output**. The Perl interpreter will run a program adding numbers and strings, but will in addition generate an error message. What does it say?

c) Arrays

Type the following program:

```
my @n = (1,2,3,4,5);
print "@n\n";
```

`@n` is an array that can hold a sequence of scalars called elements. The `print` command will print each of the elements of the array separated by spaces. Try it.

Elements of an array are index 0, 1, 2, etc. Add this line:

```
print "$n[2]\n";
```

`$n[2]` will access the third element in the array `@n`. Since the element of an array is a scalar, we can initiate a new scalar to any element in the array:

```
my $element = $n[2];
print "$element\n";
```

The index itself is also a scalar:

```
my $index = 4;
print "$n[$index]\n";
```

Try it and experiment!

A special syntax let you access the last element in an array:

```
my $last_index = $#n;
print "$n[$last_index]\n";
```

Perl operates with scalar and array contexts. In a scalar context, an array is simply the number of elements in that array. Try this:

```
my $no_elements = @n;
print "$no_elements\n";
```

d) Loops

Loops are essential in any programming language. The `foreach` loop iterates through all elements in an array using the default variable `$_`:

```
my @n = (1,2,3,4,5);
foreach (@n) {
  print "$_\n";
}
```

Try to explain what this program does!

You can rename `$_` to any scalar you like:

```
foreach my $element (@n) {
  print "$element\n";
}
```

If you want to know the index of each element, the `for` loop is more convenient:

```
for (my $i = 0; $i  < @n ; $i++) {
  print "$i: $n[$i]\n";
}
```

This `for` loop keeps on looping as long as the scalar `$i` is smaller than the number of elements in the array `@n` (i.e. `$i < @n;`). `$i` is initiated to zero (`my $i = 0;`) and is increased by one in each iteration (`$i++`).

e) Conditions

Conditions, or if-sentences, are used to control the flow of the program. Try this:

```
my @n = (1,2,3,4,5);
foreach (@n) {
  if ($_ == 4) {
    print "The array contains a 4\n";
  }
}
```

You can abort a loop using `last`:

```
foreach (@n) {
  if ($_ == 4) {
    print "The array contains a 4\n";
    last;
  }
}
```

What is the principle difference between this program and the one that did not use `last`?

Task 2 - Sorting

Now that you have seen how some of the core commands in Perl works, let's try to write a real program. Implement in Perl the sorting algorithm below. Your job is to translate the generic pseudo-code into working Perl code. Each line in the pseudo-code can be translated into one line of Perl code.

```
1       array ← list of numbers
2       n ← length of array
3       for i ← 1 to n-1
4               index ← i
5               for k ← i + 1 to n
6                       if array_k < array_index
7                               index ← k
8               Swap elements a_i and a_k
```

Task 3 - Some small programs

Implement this pseudo-code in Perl:

a) Quadratic equation: Solves a quadratic equation of the form $ax^2 + bx + c$

```
1   (a,b,c) ← three numbers
2   root ← b^2-4ac;
3   if root < 0
4           print "No solution!"
5   x1 ← (−b+√root) / 2a
6   x2 ← (−b−√root) / 2a
```

b) Remove duplicates: Removes duplicates in a list

```
1   list ← list of numbers
2   n ← length of list
3   newlist ← ()
4   for i ← 1 to n
5       m ←length of newlist
6       foundDuplicate ← false
7       for j ← 1 to m
8           if list_i = newlist_j
9                   foundDuplicate = true
10                  break
11      if foundDuplicate = false
12          add list_i to newlist
```

c) **Optional (advanced):** Count: Counts from (*0, 0, …, 0*) to **n** = (n_1, n_2, …, n_m)

```
1   c ←(0, 0, …, 0)
2   while forever
3       for i ←m to 1
4           if c_i = n_i
5               c_i ←0
6           else
7               c_i ← c_i + 1
8               break
9       print c
10      if c = (0, 0, …, 0)
11          break
```

Task 4 - Reading/writing to file

Write a Perl program that reads in the numbers in *numbers.txt* and compute the average.

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