Perl TD1
Lists and Arrays

Summary

Don't forget the perl documentation with the command *perldoc -f cmd.*

A *list* represents ordered scalar data, an array is a variable that contains a list.

```
# Assign a list to an array variable:
my @tab = (9,14,8);
my @cop = @tab; # array copy
# Access to an element of the array
my $b = $tab[0]; # 1st element of the array
$tab[1] = 5;
my $a = $tab[$#tab]; # last element
( index: size-1 of the array )
# Array traveral:
for (my $i=0; $i<=$#tab; $i++){    OR   for my $elem (@tab) {
    print $tab[$i], " ";                   print $elem, " ";
}                                      }
```

split et **join** functions : characters (or regular expressions) can be used to split a string into fields. The split function performs this task, while the join function glues the pieces together again :

```
my @fields = split; # cuts $_according to the spaces :
                # equivalent to split(" ", $_)
@fields = split(":", $line); # cuts $line according to the ':'
my $result = join("+",@fields); # "$fields[0]+$fields[1]+..."
$result = join("+","x",@fields); # "x+$fields[0]+$fields[1]+..."
$result = join("+",@fields,"x"); # "$fields[0]+$fields[1]+...+x"
```

push et **pop** functions : to add and remove items on the right side of the list :

```
my @tab = (1,2,3);
push(@tab,4) # 1 2 3 4
my $b = pop(@tab); # 1 2 3 et $b=4
push(@tab,5,6,7) # 1 2 3 5 6 7
```

shift et **unshift** functions : idem on the left side.

Reading the input (the standard input or the file given when the script is called) line by line :

```
while (<>) {
      chomp; #deletes the return carriage at the end of each line
      print ‘‘I saw $_ in input. \n’’;
}
```

1 Excercice 1

Write a Perl script that reverses the order of the elements of an array.
Hints : You can use the push, pop, shift, unshift functions.

2 Excercice 2

Write a Perl script that reads a nucleotide sequence and for each base A, C, G and T displays how many there are in the given sequence.

3 Excercice 3

Write a Perl script that does the following things :

1. Define an array of oligonucleotide sequences :

   ```
   @seq = ("tcgtgccca", "tgtt", "cccga", "ttcatcag",
   "ggcaag", "ctg", "ggtgtaccggtgatcac",
   "ccaccta", "cctgaattat" );  Ecrivez un script Perl qui
   renverse l’ordre des éléments d’un tableau. Indices
   :on pourra utiliser les fonctions push, pop, shift,
   unshift. # OU (pour eviter les apostrophes)
   # @seq = qw(tcgtgccca tgtt cccga ttcatcag ggcaag ctg \
   # ggtgtaccggtgatcac ccaccta cctgaattat );
   ```

2. Divide the elements of the @seq array into 3 arrays : @small_seq will contain oligos with between 0 and 5 bases, @medium_only those with between 6 and 10 bases, and @large only those with 11 bases and more.
3. Display each of the resulting arrays.
4. Find the longest oligo in @seq and display it.

   Hints : the operator **length** returns the length of a chain, the operator **push** allows you to add an element to an array, and the operator **join** is very useful for displaying simply (in a single instruction) the complete content of a table.

4 Exercice 4

Now suppose that your Perl script, instead of defining an array of oligonucleotide sequences, reads them in a file. The file format is simple : one oligo per line. Modify your script so that it is able to read any number of oligo sequences in a file and do the same analysis as in the exercise 3.

5 Exercice 5

Write a Perl script that recognizes the phone numbers of the form 05-40-00-66-69 or +33-5-40-00-00-66-69. Your program must be able to read any text file and display all phone numbers that it contains. The file may contain zero, one or more phone numbers per line.

Hints : Cut each line with the operator **split**, store the result in an array, then process each of the elements of the array.