Master de Bioinformatique **Perl**
Universités Bordeaux 1 & 2 **Avril 2006**

Perl - Partie 3
Listes et Tableaux

Résumé. Une *liste* représente des données scalaires ordonnées, un *tableau* est une variable qui contient une liste.

```
# Assigner une liste à une variable tableau :
     @tab = (9,14,8);
     @cop = @tab;    # recopie de tableau
# Accès à un élément du tableau :
     $b = $tab[0];   # 1ère case
     $tab[1] = 5;
     $a = $tab[$#tab];  # dernière case, ie taille-1 du tableau
# Parcours du tableau :
     for ($i=0 ; $i<=$#tab ; $i++) {    OU     foreach $elem (@tab) {
       print $tab[$i], " ";                      print $elem, " ";
     }                                         }
```

Tableau spécial : `@_` contient la liste des arguments d'un sous-programme (cf. mips4.pl). Pour accèder à sa 1ère case : `$_[0]` (rien à voir avec la variable `$_`).

Fonctions `split` et `join` : on peut utiliser des expressions régulières pour diviser une chaîne en champs. La fonction `split` remplit cette tâche, tandis que la fonction `join` recolle les morceaux.

```
@fields = split;  # découpe $_ selon les blancs
                  # équivalent à split(/\s+/,$_)
@fields = split(/:/,$line);  # découpe $line selon les :

$result = join("+",@fields);     #  "$fields[0]+$fields[1]+..."
$result = join("+","",@fields);  # "+$fields[0]+$fields[1]+..."
$result = join("+",@fields,"");  #  "$fields[0]+$fields[1]+...+"
```

Fonctions `push` et `pop` : pour ajouter et enlever des éléments côté droit de la liste.

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

Fonctions `shift` et `unshift` : idem `push` et `pop` côté gauche.

Exercice 1 Écrivez un script Perl qui réalise les choses suivantes :

1. Définissez un tableau de séquences d'oligonucléotides :

```
@seq = ("tcgtgccca", "tgtt", "cccga", "ttcatcag", "ggcaag",
        "ctg", "ggtgtaccggtgatcac", "ccaccta", "cctgaattaat");
```

   OU (pour éviter de mettre les apostrophes)

```
@seq = qw(
          tcgtgcca
          tgtt
          cccga
          ttcatcag
          ggcaag
          ctg
          ggtgtaccgtgatcac
          ccaccta
          cctgaattaat
         );
```

2. Répartissez les éléments du tableau `@seq` en trois tableaux : `@small_seq` contiendra les oligos ayant entre 0 et 5 bases (caractères), `@medium_seq` ceux ayant entre 6 et 10 bases, et `@large_seq` ceux ayant 11 bases et plus.
3. Affichez chacun des tableaux résultants.
4. Trouvez le plus long oligo dans `@seq` et affichez le.

Indice : L'opérateur `length` retourne la longueur d'une chaîne, l'opérateur `push` permet d'ajouter un élément à un tableau, et l'opérateur `join` est très utile pour afficher simplement (en une instruction) le contenu complet d'un tableau.

Exercice 2 Supposons maintenant que votre script Perl, à la place de définir un tableau de séquences d'oligonucléotides, les lise dans un fichier. Le format du fichier est très simple : un oligo par ligne. Ainsi, le fichier contenant le tableau de séquences de l'Exercice 1 est :

```
tcgtgccca
tgtt
cccga
ttcatcag
ggcaag
ctg
ggtgtaccggtgatcac
ccaccta
cctgaattaat
```

Modifiez votre script pour qu'il soit capable de lire un nombre quelconque de séquences d'oligonucléotides dans un fichier et de faire la même analyse que dans l'Exercice 1.

Exercice 3 Écrivez un script Perl qui reconnait les numéros de téléphone de la forme

`05-56-84-66-69` ou `+33-5-56-84-66-69`

Votre programme doit être capable de lire un fichier texte quelconque et afficher tous les numéros de téléphone qu'il contient. Le fichier peut bien entendu contenir zéro, un ou plusieurs numéros de téléphone par ligne.
Indice : découpez chaque ligne avec l'opérateur `split` pour la stocker dans un tableau, puis traitez chacun des éléments du tableau.