DEUXIEME PARTIE

1/ Les variables de type tableau
2/ Les opérateurs **split** and **join**
3/ Manipulation des expressions régulières pour le filtrage des motifs
4/ Ouverture de fichiers en lecture/écriture ...

1/ VARIABLES DE TYPE TABLEAU

IL existe trois types de variables en Perl :
- scalaires = variables avec une valeur singulière, un seul élément
les deux autres types de variable sont des "collections" d'éléments
- tableaux (ou tableaux listes)
- tableaux associatifs (ou tables de hachage)

Regardons plus en détail le type **tableau**.

- Un tableau représente une liste ordonnée de valeurs scalaires ; chaque élément d'un tableau est une variable scalaire.
Les noms de variables de type tableau sont précédés par le symbole @.

Par exemple, on peut stocker une ligne de fichier dans un tableau afin d'effectuer certaines opérations dessus:
@ligne=("the", "cat", "is", "on", "the", "mat", ".");
On peut se représenter cette variable comme ceci :

The	Cat	is	on	the	mat	.
0	1	2	3	4	5	6

Chaque élément du tableau est placé dans une case du tableau (une zone mémoire associée) à laquelle on peut accéder à l'aide de l'indice de la case. NOTE : les indices dans un tableau commencent à 0. Par exemple, si on vert accéder au 5ème mot de la phrase (qui aura donc l'indice 4) on va utiliser la notation :
$ligne(4)
La valeur de cette variable scalaire (chaîne de caractères) vaut "the".
Si on passe à la ligne suivante d'un fichier, la taille et la valeur d'une variable tableau change :
@ligne=("the", "cat", "is", "sick", "and", "is", "waiting", "to", "get", "better", ".");
$ligne(4) vaut maintenant "and".

Prenons pour illustration le script suivant :

EXEMPLE 3 : Variables tableaux

```
#!/usr/bin/perl
@ligne = ("the", "cat", "is", "on", "the", "mat", ".");
print "les elements de la liste \@ligne sont : @ligne\n";
print "le premier element de la ligne est $ligne[0]\n";
print "le deuxieme element de la ligne est $ligne[1]\n";
print "le dernier indice de la ligne est $#ligne\n";
print "le dernier mot de la ligne est $ligne[$#ligne]\n";
print "la ligne comporte ", scalar(@ligne), " elements\n";
```

```
$res="la ligne comporte ".scalar(@ligne)." elements\n";
print $res; print"=====================================\n"; print "maintenant on change la valeur de la variable tableau ligne : \n";
print"=====================================\n";
@ligne=("the", "cat", "is", "sick", "and", "is", "waiting", "to", "get", "better", ".");
print "maintenant elements de la liste : @ligne\n";
print "maintenant le premier element le la phrase est $ligne[0]\n";
print "maintenant la ligne comporte ", scalar(@ligne), " elements\n";
print "maintenant le dernier indice du tableau est : $#ligne \n";
==========================
```

Le script montre :

- une façon de donner des valeurs a une variable de type tableau
- comment on peut accéder aux éléments stockes dans les cases d'un tableau en fonction de leur index. Remarquer comment, pour faire référence à un seul élément d'un tableau on passe au contexte scalaire et on utilise le symbole $.
- comment connaître le nombre d'éléments (de cases) dans un tableau - c'est à dire sa longueur : scalar(@ligne). Cette fonction est similaire à celle qu'on utilisait pour connaître la longueur d'un scalaire de type chaîne de caractères : length($ligne);
- comment connaître le dernier indice dans un tableau (qui, étant donné que les indices commencent à 0, sera inférieur de 1 à la longueur du tableau).

NOTE : un variable scalaire $ligne et le tableau @ligne n'ont rien en commun à part le nom et que, imprimés, elle vont donner le même résultat.

```
$ligne = "the cat is on the mat";
@ligne = ("the", "cat", "is", "on", "the", "mat", ".");
```

L'index d'un élément d'un tableau peut être une variable scalaire - un entier positif. Essayons le script suivant :

EXEMPLE 4 :

```
#!/usr/bin/perl
@ligne = ("the", "cat", "is", "on", "the", "mat", ".");
print "quel mot voulez-vous connaître? \n";
$indice=<STDIN> ;
print "*$indice*\n";
chomp $indice;
print "*$indice*\n";
print "vous avez demande le mot avec l'index $indice,c'est a dire le ", $indice+1,"eme mot qui est $ligne[$indice]\n";
```

EXERCICE 3 :

Modifier le petit script précédent de façon a vérifier si la phase contient assez de mots pour pouvoir afficher celui qui nous intéresse.

On peut aussi récupérer une **tranche de tableau** :

EXEMPLE 5 :

```
#!/usr/bin/perl
@ligne = ("the", "cat", "is", "on", "the", "mat", ".");
print "les trois premiers mots  sont : @ligne[0..2]\n";
```

print "les premier, troisieme et cinquieme elements de la ligne sont : @ligne[0,2,4]\n";
Remarque : etant donne qu'une tranche de tableau est aussi un tableau, pour y faire référence, on utilise le symbole @.

Traiter les éléments des tableaux

Comment faire pour traiter chaque élément d'un tableau ? Plusieurs solutions : des boucles for ou while ou la construction **foreach :**

EXEMPLE 6 :

```
#!/usr/bin/perl
@ligne = ("the", "cat", "is", "on", "the", "mat", ".");

for ($i=0; $i <= $#ligne; $i++){
   print "mot numero", $i+1, ": ",$ligne[$i], "\n";
}
```

L'exemple illustre les 3 parties constituant une boucle for :

- initialisation, ($i=0)
- condition d'arrêt ($i < $#ligne)
- modification ($i++)

A la dernière répétition de la boucle, $i sera supérieur de 1 au dernier index du tableau de mots, donc la boucle s'arrête.

Variante :

```
#!/usr/bin/perl
@ligne = ("the", "cat", "is", "on", "the", "mat", ".");
$i=0;
while ($i <= $#ligne){
   print "mot numero", $i+1, ": ",$ligne[$i], "\n";
   $i++;
}
```

L'opérateur **foreach** itère sur chaque élément des tableaux :

```
foreach $mot (@ligne){
        print "$mot \n";
}
```

On peut aussi éviter de nommer une variable pour stocker chaque élément d'un tableau en utilisant la variable spéciale $_ (celle qui nous aidait la dernière fois à désigner la ligne courante :

```
foreach (@ligne) {
        print "$_\n";
}
```

Opérations sur les tableaux

On dispose de plusieurs opérateurs pour modifier le contenu d'un tableau:

- push : rajoute une valeur à la fin d'un tableau
- pop : enlève et retourne la valeur à la fin d'un tableau

- unshift : rajoute une valeur au début d'un tableau
- shift : enlève et retourne une valeur au début d'un tableau
- splice : permet d'effacer ou ajouter des éléments dans un tableau

EXEMPLE 7 :

```
#!/usr/bin/perl
@ligne = ("the", "cat", "is", "on", "the", "mat", ".");

print "la ligne est : @ligne \n";
$dernier_mot=pop(@ligne);
print "$dernier_mot\n";

print "la ligne vaut maintenant @ligne\n";
push (@ligne, "since", "yesterday", ".");
print "la ligne vaut maintenant : @ligne\n";

unshift (@ligne, "he", "told", "me");
print "la ligne vaut maintenant : @ligne\n";

$premier_mot = shift (@ligne);
print "la ligne vaut maintenant : @ligne\n";
unshift (@ligne, "she");
print "la ligne vaut maintenant : @ligne\n";

@middle=splice(@ligne, 2,2);
print "@middle\n";
print "la ligne vaut maintenant : @ligne\n";

splice (@ligne,2,0, @middle);
print "la ligne vaut maintenant : @ligne\n";
```

On peut stocker chaque ligne d'un fichier dans la case d'un tableau comme ceci :

EXEMPLE 8 :

```
#!/usr/bin/perl
$fich=shift ;
open(FICHIER, $fich) || die "Impossible d'ouvrir le fichier : $fich\n";
while (<FICHIER>) {
  tr/A-ZÂÀÄÉÈÊËÎÏÔÖÓÛÙÜÇÑ/a-zâàäéèêëîïôöóûùüçñ/;
    push (@lignes, $_);
}
print $lignes[3];  # ceci va afficher la quatrième ligne du ficher $fich
```

Cet exemple montre une façon de construire un tableau en rajoutant chaque ligne qu'on lit en entrée dans un tableau, après avoir transforme les majuscules en minuscules. Sans effectuer aucune modification sur la ligne de commande, on peut construire un tableau des lignes d'un fichier comme ceci :

```
@lignes=<FICHIER> ;
```

Le script suivant permet d'arranger de mélanger les lignes d'un texte (les afficher dans un ordre aléatoire) :

EXEMPLE 9 :

```
#!/usr/bin/perl
$fich=shift ;
open(FICHIER, $fich) || die "Impossible d'ouvrir le fichier : $fich\n";

@lignes=<FICHIER> ;
srand;
while ($#lignes > 0){
   $index=rand($#lignes);
   print "$lignes[$index]";
   splice (@lignes, $index, 1);
}
```

L'instruction srand initialise un générateur de nombre aléatoires.
La fonction rand génère un nombre aléatoire entre 0 et le dernier indice dans le tableau, puis l'instruction print imprime la ligne stockée à la case du tableau correspondant à cet indice choisi au hasard. Enfin, l'opérateur splice élimine la case du tableau qu'on vient d'imprimer. Par conséquent la longueur du tableau diminue de 1.

EXERCICE 4 : Comment modifier le script précédent pour imprimer juste 3 lignes du corpus choisies au hasard?

2/ Les opérateurs split et join

Lors de la dernière séance on a étudié les variables scalaires de type chaîne de caractères. On a vu lors de la dernière séance comment on peut concaténer des chaînes de caractères à l'aide de l'opérateur .
Le même résultat pourrait être obtenu à l'aide de l'opérateur **join** qui permet en plus de définir un séparateur à utiliser entre les chaînes de caractères qu'on concatène :

EXEMPLE 10 :

```
#!/usr/bin/perl
$word="the";
$cat="Det";
$lemma="the";
$index=0;
$entree=join (';', $word, $cat, $lemma, $index);
print "$entree\n";
```

Le premier argument de l'opérateur join est le séparateur, suivi par une liste de variables à concaténer. L'opérateur peut s'appliquer aussi à un tableau :

EXEMPLE 11 :

```
#!/usr/bin/perl
@ligne = ("the", "cat", "is", "on", "the", "mat", ".");
$ligne = join (';', @ligne);
```

print $ligne ;

L'opérateur **split** à l'effet contraire : il permet de segmenter un chaîne de caractères en fonction d'un séparateur défini :

EXEMPLE 12 :

```
#!/usr/bin/perl
$ligne = "the;cat;is;on;the;mat;.";
@ligne = split (';', $ligne);
print "@ligne\n";
```

Cet opérateur pourrait nous permettre de segmenter une ligne lue en mots (délimités par des espaces ou des tabs) et les stocker dans un tableau @ligne.

Analyser le script suivant, qui permet de segmenter un texte en n fragments comportant un nombre de mots donné :

EXEMPLE 13 :

```
#!/usr/bin/perl

$corpus = shift;
$nb = shift;

open(CORPUS, $corpus) or die "Could not open corpus file: $corpus for read\n";

@ets=("adeferra","bcone","ctournai","jhinojos","mcoux","mmedart","salfande","apilorge","clejeune","dadeikal","jtheveno","mgueguen","mnoel","striquet","apoujol","cmairet","fafkir","llheureu","mhy","njackson");


$i=0;
$j=0;
print "\n=====================================\n     CE QUI SUIT EST LA PART DE $ets[$j++] \n=====================================\n";
while(<CORPUS>)
{

    @ligne=split ($_, / \t/;
    foreach $mot (@ligne){
          print "$mot ";
          $i++;
          if ($i==$nb){
         # if ($i==1185){
             print "\n============================\n     CE QUI SUIT EST LA PART DE $ets[$j++]\=========================\n";
             $i=0 ;
          }
```

```
    }

}
```

3/Ouverture de fichiers en lecture ou écriture

Dans cette section on s'intéresse aux possibilités que Perl offre pour ouvrir un fichier - lire dans le ficher -ou écrire dans un fichier.

Dans les exemples précédents on a vu comment ouvrir un fichier avec la ligne de commande :

open (FICH, $file) or die "Impossible d'ouvrir le fichier $file! \n";

$file est une variable scalaire de type chaîne de caractères qui correspond au nom du fichier qu'on veut traiter
FICH est un descripteur, dont on va se servir par la suite pour faire référence au fichier.
Si le fichier n'existe pas ou si l'on se trompe sur le nom du fichier l'exécution du programme s'arrête et le programme affiche le message d'erreur.

Dans les exemples précédents on a vu également comment lire (parcourir un fichier ligne par ligne) à l'aide de l'opérateur <> qui permet de lire, à chaque exécution de la boucle while un bout de texte allant jusqu'au retour chariot. (\n). La ligne lue à chaque itération de la boucle while est stockée dans la variable prédéfinie $_.

Ce comportement peut être modifié en modifiant la variable spéciale $/.
On peut, en changeant la valeur de cette variable lire dans un fichier soit paragraphe par paragraphe, soit jusqu'à trouver un certain délimiteur.

EXEMPLE 14 :

```
#!/usr/bin/perl  # EN.P
$fich=shift ;
open(FICHIER, $fich) || die "Impossible d'ouvrir le fichier : $fich\n";
$/="\n\n";

@lignes=<FICHIER> ;
print "@lignes[2]";
```

Autre exemple : Corpus.petit

EXEMPLE 15 :

```
#!/usr/bin/perl
$fich=shift ;
open(FICHIER, $fich) || die "Impossible d'ouvrir le fichier : $fich\n";
$/="</s>\n";

@lignes=<FICHIER> ;
print "@lignes[2]";
```

Comment écrire dans un fichier? Étudier le script suivant :

EXEMPLE 16 :

```
#!/usr/bin/perl
$fich=shift ;
$petites="petites.txt";
$longues="longues.txt";

open(FICHIER, $fich) or die "Impossible d'ouvrir le fichier : $fich\n";
open(PETITES, ">$petites") or die "Impossible d'ouvrir le fichier : $petites pour l'ecriture \n";
open(LONGUES, ">$longues") or die "Impossible d'ouvrir le fichier : $longues pour
l'ecriture \n";
while (<FICHIER>){
   if ((length ($_) < 40) && (length ($_)>1)){
         print PETITES $_;
   }else{
         print LONGUES $_;
   }
}
close (FICHIER);
close (PETITES);
close (LONGUES);
```

Si on applique le script a plusieurs fichiers, le contenu de fichiers petites.txt et longues.txt est écrasé à chaque fois.

Si on veut éviter, et juste rajouter des lignes a la fin de deux fichiers, il suffit d'ouvrir les deux fichiers en mode "append" :

```
open(LONGUES, ">>$longues") or die "Impossible d'ouvrir le fichier : $longues pour
l'écriture \n";
```

4/ Manipulation des expressions régulières

En lisant un fichier ligne par ligne on peut vouloir appliquer un traitement spécial à des lignes comportant un certain motif.

EXEMPLE 17 :

```
#!/usr/bin/perl
# exemple recette.html
$fich=shift ;
open(FICHIER, $fich) || die "Impossible d'ouvrir le fichier : $fich\n";

while (<FICHIER>) {
   if($_ =~ /www/){
         print $_;
   }
}
```

Ce programme permet d'afficher les lignes d'un fichier comportant le motif www. D'autre part, l'opérateur !~ permet d'exclure les lignes comportant un certain motif : if($_ !~ /www/)

Soit le fichier glossaire.txt, dont les lignes sont constituées d'un terme français suivi par le signe : , suivi par son équivalent en anglais comme dans les lignes suivantes :

```
Abc&egrave;s : Abscess
Abc&egrave;s chaud : Acute abcess
Abc&egrave;s froid : Cold abscess
Absorption dermique : Dermal uptake
Absorption percutan&eacute;e : Cutaneous absorption
Acanthok&eacute;ratolyse : Acanthokeratolysis
Acantholyse : Acantholysis
Acanthome : Acanthoma
Acanthome fissur&eacute; : Acanthoma fissuratum
Acanthose : Acanthosis
```

Comment peut-on isoler les deux parties d'une part le terme anglais, d'autre part le terme français et les afficher dans l'ordre inverse ?
Une méthode consisterait à utiliser l'opérateur split avec le délimiteur " : ". Comme ceci :

EXEMPLE 18 :

```
#!/usr/bin/perl
$fich=shift ;
open(FICHIER, $fich) or die "Impossible d'ouvrir le fichier : $fich\n";
while (<FICHIER>){
   ($ch1,$ch2)=split(/ : /, $_);
   print "$ch2 xxxx $ch1 ";  # un chomp nécessaire ici?
}
close (FICHIER);
```

(Dans les exemples précédents d'utilisation de l'opérateur split, on stockait le résultat dans un tableau; dans cet exemple, étant donné qu'on connaît le nombre de champs sur chaque ligne su ficher, on peut stocker les deux dans 2 variables scalaires).

Une autre méthode serait d'isoler les deux champs à l'aide d'une expression régulière, puis les réutiliser en utilisant les variables prédéfinies $1 et $2 comme ceci :

EXEMPLE 19 :

```
#!/usr/bin/perl

$fich=shift ;
open(FICHIER, $fich) or die "Impossible d'ouvrir le fichier : $fich\n";
while (<FICHIER>){
   chomp $_;
   $_=~/^(.+) : (.+)$/;
   $french=$1;
   $english=$2;
   print "$english xxx $french\n";
}
close (FICHIER);
```

Greedy vs stingy matching en Perl

(ou des filtrages de motifs gloutons ou restrictifs)

EXEMPLE : Soit le fichier titi.txt qui ne contient que la ligne suivante :

<I> ce texte </I> est en <I> italiques </I> et <I> je</I> veux extraire les parties en italique .

Comment peut-on récupérer les bouts de texte compris entre balises ?
Etant donné qu'on connaît la taille du fichier, on n'a pas besoin d'utiliser une boucle afin de parcourrir le fichier, mais on pourra lire juste cette ligne et la stoker dans une variable comme ceci :

$ligne = <FICHIER>

Questions :
1/ qu'affichera à l'écran le script suivant appliqué à titi.txt? (Autrement dit, que vaut la variable $1 dans ce cas?)

```
#!/usr/bin/perl
$fich=shift ;
open(FICHIER, $fich) || die "Impossible d'ouvrir le fichier : $fich\n";

$ligne = <FICHIER>);
if ($ligne =~/<I>(.*)<\/I>/){
      print $1;
}
```

2/ Et le script suivant ? (Ligne modifiée en gras)

```
#!/usr/bin/perl
$fich=shift ;
open(FICHIER, $fich) || die "Impossible d'ouvrir le fichier : $fich\n";

$ligne = <FICHIER>);
if ($ligne =~/<I>(.*?)<\/I>/){
      print $1;
}
```

3/ Ou alors le suivant :

```
#!/usr/bin/perl
$fich=shift ;
open(FICHIER, $fich) || die "Impossible d'ouvrir le fichier : $fich\n";

$ligne = <FICHIER>);
if ($ligne =~/<I>(.*)<\/I>/i){
      print $1;
}
```

4/ Enfin le suivant :

```
#!/usr/bin/perl
$fich=shift ;
open(FICHIER, $fich) || die "Impossible d'ouvrir le fichier : $fich\n";

while (<FICHIER>) {
    chomp $_;
```

```
    @trouves = /<I>(.*?)<\/I>/ig;
    print "premier trouv□é : $trouves[0]\n";
    print "deuxieme trouv□é : $trouves[1]\n";
    print "troisieme trouv□é : $trouves[2]\n";
    print "quatreme trouv□é : $trouves[3]\n";
}
```

Observation :
Par défaut, Perl effectue un filtrage de motifs (*pattern matching*) dit 'gloutton'. Dans l'exemple 1 ci-dessus, une fois la balise ouvrante <I> trouvée, on repart à la fin de la ligne en quête de la balise fermante </I>. Ceci fait que le texte récupéré entre les deux balises peut contenir des balises.

Pour déconnecter ce fonctionnement par défaut de Perl, dans l'exemple 2 on a introduit un signe ? après l'expression comprise entre balises. Ceci déclenche un filtrage de motifs dit restrictif. Cette fois-ci, on cherche le plus petit bout de texte correspondant à la requête.

Dans le deux cas, on a cherche uniquement les balises avec un I en majuscule. Si l'on souhaite ignorer la différence entre les balises <I> et <i>, il suffit d'ajouter l'option i comme dans l'exemple 3 ci-dessus :
`if ($ligne =~/<I>(.*)<\/I>/i){`

Enfin, si plusieurs bouts de texte en italique apparaissent dans la ligne, on est obligés de les stoker dans un tableau d'éléments trouvés, qu'on affiche case par case dans l'exemple 4.
RAPPEL : lors de l'éude de la commande `grep`, on utilisait l'artifice suivant pour simuler un filtrage de motif restrictif :

```
grep –E  "<I>[^<]+</I> " texte.html
```

A la différence de "<I>.+</I>", l'expression ci-dessus nous permet de faire une requête restrictive.

EXERCICE : soit le fichier corpus.shtml (http://www.eila.jussieu.fr/~avolansk/COURS-IL/corpus.shtml). Ecrire le script enlève_balises.pl permettant de transformer le fichier en texte en remplaçant les balises de type <li>, marque des listes à puces, par des tirets.
EXERCICE : transformer le fichier html vocabulaire_anglais.html en un fichier xml valide de la forme :

```
"<?xml version=\"1.0\" encoding =\"ISO-8859-15\"?>\
<document>
      <entree id="1">
            <anglais>dry-stone</anglais>
            <francais>construction en pierre sèche</francais>
      </entree>
      <entree id="2">
      ...
      ...
</document>
```

```
On notera la forme des lignes qui nous intéressent :
```
dry-stone/dry-masonry constructionla construction en pierre sèche