Perl TD4 : Tableaux associatifs (ou hachages)

Résumé sur les tableaux associatifs

Les tableaux associatifs (ou hachages), comme les tableaux classiques, sont une collection de données, mais les indices repérant les éléments ne sont pas forcément des entiers, mais des valeurs quelconques que l'on nomme clés. L'ordre des couples (clé/élément) dans un hachage est imprévisible. Il ne dépend que de la manière dont Perl les stocke pour faciliter sa recherche en fonction des clés. De plus, l'accès à un hash inexistant implique sa création. Le nom d'une variable de type hachage commence par % (Rq : les variables %hash, @hash et $hash n'ont rien à voir entre elles).

```
# Assigner un element (crée le hachage s'il n'existe pas)
my %hash;
$hach{"aaa"} = 3; # cle = "aaa", valeur = 3
$hach{5.6} = "abc"; # cle = "5.6", valeur = "abc"
# Initialisation globale ou copie de hachage
%hach = ("aaa",3,5.6,"abc"); # idem que ci-dessus
#ou encore
%hach = (aaa => 3, 5.6 => abc); # idem que ci-dessus
%cop = %hach; # recopie de hachage
```

Fonction **keys** : fournit la liste de toutes les clés d'un hachage. Utilisée avec une boucle foreach, elle permet de parcourir tous les éléments du hachage (sans bien sûr maîtriser l'ordre du parcours) :

```
@liste = keys(%hach); # @liste = ("aaa", 5.6) ou (5.6, "aaa")
foreach $cle (keys(%hach)) {
    print "cle: $cle, valeur: $hach{$cle}\n";
}
```

Fonction **values** : fournit la liste de tous les éléments d'un hachage, dans le même ordre que la liste des clés renvoyées par keys :

```
@liste = values(%hach); # @liste = (3,"abc") ou ("abc",3)
```

Fonction **each** : fournit un couple (clé/valeur) sous forme d'une liste à 2 elé-ments. A chaque application sur le même hachage, le couple suivant est renvoyée (liste vide si c'est fini) :

```
while ( ($cle,$val)=each(%hach) ) {
    print "cle: $cle, valeur: $val\n";
}
```

Fonction **delete** : permet de supprimer un élément d'un hachage :

```
delete $hach{"aaa"}; # %hach ne contient plus le couple cle/element "aaa"/3
```

Résumé sur les fichiers

Un descripteur (ou handle) de fichier est le nom d'une connexion d'E/S (Entrée/Sortie) etabli entre un programme Perl actif et le monde extérieur. Perl fournit 3 descripteurs de fichiers qui s'ouvrent chacun automatiquement sur une E/S précise :

— STDIN : entrée standard,
— STDOUT : sortie standard,
— STDERR : sortie d'erreurs standard.

Pour ouvrir tout autre descripteur de fichier, utiliser la commande open() :

```
open (FICDESC1, ‘‘nom_fichier’’); # ouvrir en lecture
open (FICDESC2, ‘‘>nom_fichier’’) # ouvrir en ecriture
open (FICDESC3, ‘‘>>nom_fichier’’) # ouvrir en ajout
```

Pour fermer un descripteur de fichier :

```
close (FICDESC);
```

L'utilisation est très semblable a ‘ <>’. Exemple :

```
open (EP, ‘‘/etc/passwd’’);
open (RES, ‘‘>res.txt’’);
while (<EP>) {
    chomp;
    print RES ‘‘I saw $_ in passwd file.\n’’;
}
close (EP);
close(RES);
```

L'opérateur die() prend une liste optionnelle de paramètres, l'affiche (à la façon de print) sur STDERR et termine le programme Perl. Exemple :

```
open (DATA, ‘‘~/perl/rapport.txt’’) || die ‘‘Fichier ~/perl/rapport.txt n’existe pas.\n’’;
```

OU

```
#La variable $! contient la chaîne d'erreur décrivant l'erreur système
#la plus récente.
open (DATA, ``~/perl/rapport.txt'') || die ``ouverture impossible: $!.\n'';
```

Petit résumé expressions régulières

Ce qui suit n'est pas du tout exhaustif. Pour plus d'informations et des exemples, reportez- vous à la documentation de Perl.

Trouver, extraire, remplacer une sous-chaîne :

```
#index retourne la position du premier caaractère de $souschaine dans
#$chaine, retourne -1 si non trouvé. Possibilité de rajouter un 3e
#paramètre indiquant l'indice dans $chaine à partir duquel
#on recherche $souschaine
$pos = index($chaine, $souschaine);
#substr extrait toute la sous-chaîne de taille $longueur à partir de
#la position $debut dans $chaine
$ch = substr($chaine,$debut,$longueur);
#substr extrait toute la fin de $chaine à partir de la position $debut
$ch = substr($chaine,$debut);
#substr remplace la sous-chaîne de taille $longueur à partir de
#la position $debut dans $chaine par $nouvsouschaine.
substr($chaine,$debut,$longueur) = $nouvsouschaine;
```

Avec des expressions régulières

```
/$exprreg/ #opérateur d'extraction
s/$exprreg/$chaine/ #opérateur de substitution
tr/$caractere/$Caracteres/ #opérateur de transcription
```

Ces trois opérateurs s'appliquent par défaut à la variable $_. Si on veut les appliquer à d'autres variables que $_, il faut utiliser l'opérateur '=~'. Quelques exemples supplémentaires :

```
#change tous les a en b et tous les b en a
tr/ab/ba/;
#les lettres a, b, c et d deviennent respectivement A, B, C et D,
#et toutes les lettres de e à z deviennent toutes D
tr/a-z/ABCD/;
#les lettres a, b, c et d deviennent respectivement A, B, C et D,
#et toutes les lettres de e à z sont supprimées
tr/a-z/ABCD/d;
#compte le nombre de substitutions sans modifier $_
$count = s/..././/;
#compte le nombre de transcriptions sans modifier $_.
$count = tr/..././/;
```

1 Préambule

Pour ce TD nous allons travailler sur le texte de Hamlet (The Tragedy of Hamlet, Prince of Denmark) de Shakespeare. Nous allons utiliser une version moderne du manuscrit décrit dans un fichier XML. Ce fichier est très facile à trouver notamment à `http://gnosis.cx/download/hamlet.xml`. Nous allons analyser ce fichier avec un analyseur écrit en Perl et calculer diverses statistiques. Les balises XML seront traités à la main dans un premier temps. L'objectif des questions suivantes est dans un premier temps de lire le fichier XML, construire une structure de données à base de tableaux associatifs et en utilisant cette structure de répondre à des questions sur la pièce. L'application à construire devra disposer d'un menu (pas d'interface graphique, uniquement en mode texte) qui proposera des options pour répondre aux différentes questions.

Questions

1. Donner le nombre de personnages, d'actes, de scènes et le nombre de scènes par actes.
2. Ajouter le nombre de personnages apparaissant dans chaque acte et chaque scène (au choix de l'utilisateur).
3. Donner le nom de chaque personnage en classant la liste par ordre alphabétique.
4. Donner le nombre de fois que chaque personnage prend la parole. Est-ce qu'il y a des rôles muets ? Quel est le personnage qui parle le plus ?
5. Donner le nombre de scènes où Hamlet parle. Où Hamlet et Ophélia parlent.
6. Quand (acte et scène) Hamlet prononce t-il la célèbre réplique : "To be, or not to be : that is the question" ?
7. Selon le temps disponible, refaire la même chose en utilisant le module `XML::Parser` (installé par défaut).