Travaux dirigés de Perl n°3/6

Entrées/sorties, références

—Université Gustave Eiffel—

Le but de ce TD est de continuer l'apprentissage du langage *Perl* par des exemples d'utilisation des fichiers et des références.
Toutes les fonctions écrites devront être testées.

▸ **Exercice 1.** ***(Fichiers)***

Écrivez un programme qui :

1. *place dans un tableau la liste des noms des entrées (fichiers, répertoires, etc) de votre répertoire personnel qui commencent par un point.*
2. *ne conserve dans ce tableau que les non-exécutables (droit UNIX 'x') (fonction* `grep`*),*
3. *trie ce tableau par ordre de taille du fichier (ne pas confondre avec la taille du nom du fichier),*
4. *place dans un second tableau la taille respective de chaque fichier du tableau (fonction* `map`*).*
5. *(optionnel) crée une table de hachage ayant pour clefs les noms de fichier et pour valeur leur taille respective.*

▸ **Exercice 2.** ***(Parcours de références)***

Construisez une référence vers une table de hachage dont les clefs seraient le nom d'une personne (prenez pour exemple au moins 3 personnes) et la valeur une référence vers une table de hachage comportant les clefs suivantes :

- `Tel` *: son numéro de téléphone,*
- `Adr` *: son adresse,*
- `Enfants` *: une référence vers un tableau des prénoms de ses enfants.*

Sur papier, faites un schéma de cette structure en mémoire.

Affichez d'abord cette structure de données à l'aide du module `Data::Dumper`

Écrivez ensuite le code Perl permettant d'afficher toute la structure sans utiliser ce module (le format d'affichage est libre) : n'utilisez pas l'opérateur `ref`*, mais utilisez le fait que vous connaissez les trois clefs des tables de hachages (*`Tel`*,* `Adr` *et* `Enfants`*) ainsi que les types de leurs valeurs respectives.*

Dans un premier temps, affichez la liste des enfants avec `foreach`*, dans un second avec* `join`*. Comment afficher le nombre d'enfants de chaque personne ?*

▸ **Exercice 3.** ***(Création de références)***

Le but de cet exercice est la manipulation du fichier `passwd`*.*

1. *Écrivez une fonction* `parse` *qui prend en paramètre le nom du fichier à analyser. Elle renvoie une référence vers une table de hachage qui aura pour clefs le login de chacun des utilisateurs et pour valeurs une référence vers une table de hachage dont les clefs seront* `passwd`*,* `uid`*,* `gid`*,* `info`*,* `home` *et* `shell`*. Elle s'utilisera de la sorte :*
   ```
   my $ref = parse( '/home/ens/lhullier/ens/passwd' );
   ```
2. *Affichez la valeur de retour de cette fonction à l'aide du module* `Data::Dumper`

3. *Écrivez une fonction* `display1` *qui effectue votre propre affichage de toutes ces informations ; affichez la table de hachage de chaque utilisateur avec une boucle.*
4. *Écrivez une fonction* `display2` *qui effectue le même affichage par ordre alphabétique des logins.*
5. *Écrivez une fonction* `display3` *qui effectue le même affichage par ordre inverse des uid (pensez au fait que vous avez accès à la référence dans le bloc de comparaison de sort). Pour les uid égaux, peut-on trier par ordre alphabétique des logins ?*

▸ **Exercice 4.** ***(Fonctionnelles)***

Le but de cet exercice est d'écrire une version simplifiée des fonctions `grep`, `map` *et* `sort` *(bien-sûr sans utiliser ces fonctions disponibles dans le langage). Vos fonctions auront les paramètres suivants :*

- *une référence vers une fonction, cette fonction effectuera le traitement spécifique et/ou retournera la valeur nécessaire à votre fonction,*
- *la liste des valeurs à traiter (ou indifféremment un tableau, étant donné l'aplatissement).*

Les exemples sont donnés sur des entiers, mais vos fonctions doivent pouvoir être utilisées sur n'importe quels scalaires.

Votre fonction `mygrep` *renverra la liste des éléments pour lesquels la fonction passée en paramètre renvoie vrai. Elle pourra être appelée de la sorte :*

```
sub positif {
   my ($e) = @_;
   return $e > 0;
}
my @t = mygrep \&positif, 43,654,-43,34,32,-23,652,1,2,1,523;
```

Votre fonction `mymap` *renverra la liste des éléments images à travers la fonction passée en paramètre de la liste argument (chacun des éléments de la liste renvoyée sera la valeur de la fonction appliquée à chaque élément de la liste de départ). On ne gérera pas le cas d'une modification de la liste initiale. Votre fonction pourra être utilisée ainsi :*

```
sub double {
   my ($e) = @_;
   return 2*$e;
}
my @t2 = mymap \&double, @t;
```

Votre fonction `mysort` *renverra la liste triée selon les comparaisons effectuées par la fonction passée en paramètre. Cette fonction prendra deux paramètres et renverra un entier : positif si son second paramètre doit être avant son premier, négatif si son premier paramètre doit être avant son second, nul s'ils sont équivalents.*

Vous vous servirez de cette fonction pour effectuer des comparaisons d'éléments deux à deux. Vous êtes libre de choisir la méthode de tri que vous souhaitez.

La fonction pourra être appelée ainsi :

```
sub croissant {
   my ($a,$b) = @_;
   return $a <=> $b;
}
my @tt = mysort \&croissant, @t2;
```

Vérifiez que vos fonctions peuvent se chaîner :

```
@tab = mysort \&croissant, mygrep \&positif, mymap \&double, @tab;
```