4.3 La solution

Avant même que Perl 5 ne pointât le bout de son nez, nous étions donc coincés avec ce fait de conception : les valeurs de tables de hachage doivent être des scalaires. Les références sont la solution à ce problème.

Une référence est une valeur scalaire qui *fait référence à* un tableau entier ou à une table de hachage entière (ou à à peu près n'importe quoi d'autre). Les noms sont une forme de référence dont vous êtes déjà familier. Pensez au Président des États-Unis D'Amérique: ce n'est qu'un tas désordonné et inutilisable de sang et d'os. Mais pour parler de lui, ou le représenter dans un programme d'ordinateur, tout ce dont vous avez besoin est le sympathique et maniable scalaire "George Bush".

Les références en Perl sont comme des noms pour les tableaux et les tables de hachage. Ce sont des noms privés et internes de Perl, vous pouvez donc être sûr qu'ils ne sont pas ambigus. Au contraire de "George Bush", une référence pointe vers une seule chose, et il est toujours possible de savoir vers quoi. Si vous avez une référence à un tableau, vous pouvez accéder au tableau complet qui est derrière. Si vous avez une référence à une table de hachage, vous pouvez accéder à la table entière. Mais la référence reste un scalaire, compact et facile à manipuler.

Vous ne pouvez pas construire une table de hachage dont les valeurs sont des tableaux : les valeurs des tables de hachage ne peuvent être que des scalaires. Nous y somme condamnés. Mais une seule référence peut pointer vers un tableau entier, et les références sont des scalaires, donc vous pouvez avoir une table de hachage de références à des tableaux, et elle agira presque comme une table de hachage de tableaux, et elle pourra servir juste comme une table de hachage de tableaux.

Nous reviendrons à notre problème de villes et de pays plus tard, après avoir introduit un peu de syntaxe pour gérer les références.

4.4 Syntaxe

Il n'y a que deux façons de construire une référence, et deux façons de l'utiliser une fois qu'on l'a.

4.4.1 Construire des références

Règle de construction 1

Si vous mettez un \ devant une variable, vous obtenez une référence à cette variable.

```
$tref = \@tableau;       # $tref contient maintenant une référence
                         # à @tableau
$href = \%hachage;       # $href contient maintenant une référence
                         # à %hachage
$sref = \$scalaire;      # $sref contient maintenant une référence
                         # à $scalaire
```

Une fois que la référence est stockée dans une variable comme $tref ou $href, vous pouvez la copier ou la stocker ailleurs comme n'importe quelle autre valeur scalaire :

```
$xy = $tref;             # $xy contient maintenant une référence
                         # à @tableau
$p[3] = $href;           # $p[3] contient maintenant une référence
                         # à  %hachage
$z = $p[3];              # $z contient maintenant une référence
                         # à  %hachage
```

Ces exemples montrent comment créer des références à des variables avec un nom. Parfois, on veut utiliser un tableau ou une table de hachage qui n'a pas de nom. C'est un peu comme pour les chaînes de caractères et les nombres : on veut être capable d'utiliser la chaîne "\n" ou le nombre 80 sans avoir à les stocker dans une variable nommée d'abord.

Règle de construction 2

`[ ELEMENTS ]` crée un nouveau tableau anonyme, et renvoie une référence à ce tableau. `{ ELEMENTS }` crée une nouvelle table de hachage anonyme et renvoie une référence à cette table.

```
$tref = [ 1, "toto", undef, 13 ];
# $tref contient maintenant une référence à un tableau

$href = { AVR => 4, AOU => 8 };
# $href contient maintenant une référence à une table de hachage
```

Les références obtenues grâce à la règle 2 sont de la même espèce que celles obtenues par la règle 1 :

```
# Ceci :
$tref = [ 1, 2, 3 ];

# Fait la même chose que cela :
@tableau = (1, 2, 3);
$tref = \@tableau;
```

La première ligne est une abréviation des deux lignes suivantes, à ceci près qu'elle ne crée pas la variable tableau superflue `@tableau`.

Si vous écrivez `[]`, vous obtenez une référence à un nouveau tableau anonyme vide. Si vous écrivez `{}`, vous obtenez une référence à un nouvelle table de hachage anonyme vide.

4.4.2 Utiliser les références

Une fois qu'on a une référence, que peut-on en faire? C'est une valeur scalaire, et nous avons vu que nous pouvons la stocker et la récupérer ensuite comme n'importe quel autre scalaire. Il y juste deux façons de plus de l'utiliser :

Règle d'utilisation 1

À la place du nom d'un tableau, vous pouvez toujours utiliser une référence à un tableau, tout simplement en la plaçant entre accolades. Par exemple, `@{$tref}` la place de `@tableau`.

En voici quelques exemples :

Tableaux :

```
@t              @{$tref}                Un tableau
reverse @t      reverse @{$tref}        Inverser un tableau
$t[3]           ${$tref}[3]             Un élément du tableau
$t[3] = 17;     ${$tref}[3] = 17        Affecter un élément
```

Sur chaque ligne, les deux expressions font la même chose. Celles de gauche travaillent sur le tableau `@t`, et celles de droite travaillent sur le tableau vers lequel pointe `$tref`; mais une fois qu'elle ont accédé au tableau sur lequel elles opèrent, elles leur font exactement la même chose.

On utilise une référence à une table de hachage `exactement` de la même façon :

```
%h              %{$href}              Une table de hachage
keys %h         keys %{$href}         Obtenir les clés de la
                                      table
$h{'bleu'}      ${$href}{'bleu'}      Un élément de la table
$h{'bleu'} = 17 ${$href}{'bleu'} = 17 Affecter un élément
```

Quelque soit ce que vous voulez faire avec une référence, vous pouvez y arriver en appliquant la **Règle d'utilisation 1**. Vous commencez par écrire votre code Perl en utilisant un vrai tableau ou une vraie table de hachage puis vous remplacez le nom du tableau ou de la table par `{$reference}`.

"Comment faire une boucle sur un tableau pour lequel je ne possède qu'une référence ?" Pour faire une boucle sur un tableau "normal" vous auriez écrit :

```
for my $element (@tableau) {
   ...
}
```

Remplaçons donc le nom du tableau (`tableau`) par la référence :

```
for my $element (@{$tref}) {
   ...
}
```

"Comment afficher le contenu d'une table de hachage dont je possède une référence ?" Tout d'abord, écrivons le code pour afficher le contenu d'une table de hachage :

```
for my $cle (keys %hachage) {
  print "$cle => $hachage{$cle}\n";
}
```

Puis remplaçons le nom de la table par la référence :

```
for my $cle (keys %${href}) {
  print "$cle => $${href}{$cle}\n";
}
```

Règle d'utilisation 2

La **Règle d'utilisation 1** est la seule réellement indispensable puisqu'elle vous permet de faire tout ce que vous voulez avec des références. Mais le chose la plus courante que l'on fait avec un tableau ou une table de hachage est d'accéder à une valeur et la notation induite par la **Règle d'utilisation 1** est lourde. Il existe donc une notation raccourcie.

`${$tref}[3]` est trop dur à lire, vous pouvez donc écrire `$tref->[3]` à la place.

`${$href}{bleu}` est trop dur à lire, vous pouvez donc écrire `$href->{bleu}` à la place.

Si `$tref` est une référence à un tableau, alors `$tref->[3]` est le quatrième élément du tableau. Ne mélangez pas ça avec `$tref[3]`, qui est le quatrième élément d'un tout autre tableau, trompeusement nommé `@tref`. `$tref` et `@tref` ne sont pas reliés, pas plus que `$truc` et `@truc`.

De la même façon, `$href->{'bleu'}` est une partie de la table de hachage vers laquelle pointe `$href`, et qui ne porte peut-être pas de nom. `$href{'bleu'}` est une partie de la table de hachage trompeusement nommée `%href`. Il est facile d'oublier le `->`, et si cela vous arrive vous obtiendrez des résultats étranges comme votre programme utilisera des tableaux et des tables de hachage issus de tableaux et de tables de hachages qui n'étaient pas ceux que vous aviez l'intention d'utiliser.

4.5 Un exemple

Voyons un rapide exemple de l'utilité de tout ceci.

D'abord, souvenez-vous que `[1, 2, 3]` construit un tableau anonyme contenant `(1, 2, 3)`, et vous renvoie une référence à ce tableau.

Maintenant, considérez

```
@t = ( [1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]
     );
```

`@t` est un tableau à trois éléments, et chacun d'entre eux est une référence à un autre tableau.

`$t[1]` est l'une de ces références. Elle pointe vers un tableau, celui contenant `(4, 5, 6)`, et puisque c'est une référence à un tableau, la `Règle d'utilisation 2` dit que nous pouvons écrire `$t[1]->[2]` pour accéder au troisième élément du tableau. `$t[1]->[2]` vaut 6. De la même façon, `$t[0]->[1]` vaut 2. Nous avons ce qui ressemble à un tableau de dimensions deux ; vous pouvez écrire `$t[LIGNE]->[COLONNE]` pour obtenir ou modifier l'élément se trouvant à une ligne et une colonne données du tableau.

Notre notation est toujours quelque peu maladroite, voici donc un raccourci de plus :

4.6 Règle de la flèche

Entre deux **indices**, la flèche est facultative.

A la place de `$t[1]->[2]`, nous pouvons écrire `$t[1][2]` ; cela veut dire la même chose. A la place de `$t[0]->[1] = 23`, nous pouvons écrire `$t[0][1] = 23` ; cela veut dire la même chose.

Maintenant on dirait vraiment un tableau à deux dimensions !

Vous pouvez voir pourquoi les flèches sont importantes. Sans elles, nous devrions écrire `${$t[1]}[2]` à la place de `$t[1][2]`. Pour les tableaux à trois dimensions, elles nous permettent d'écrire `$x[2][3][5]` à la place de l'illisible `${${$x[2]}[3]}[5]`.

4.7 Solution

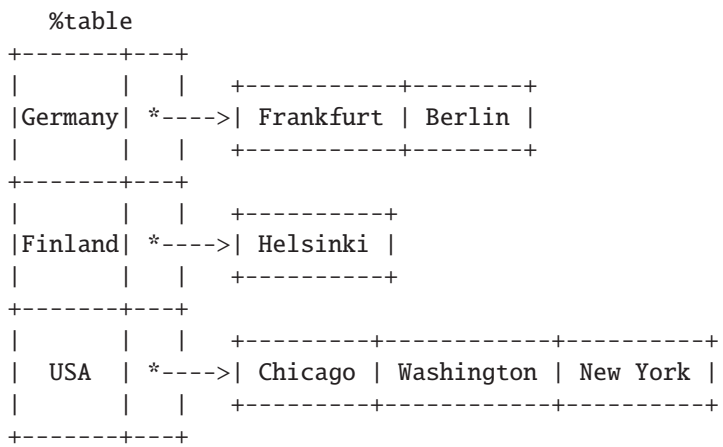
Voici maintenant la réponse au problème que j'ai posé plus haut, qui consistait à reformater un fichier de villes et de pays.

```
 1  my %table;

 2  while (<>) {
 3    chomp;
 4    my ($ville, $pays) = split /, /;
 5    $table{$pays} = [] unless exists $table{$pays};
 6    push @{$table{$pays}}, $ville;
 7  }

 8  foreach $pays (sort keys %table) {
 9    print "$pays: ";
10    my @villes = @{$table{$pays}};
11    print join ', ', sort @villes;
12    print ".\n";
13  }
```

Ce programme est constitué de deux parties : les lignes 2 à 7 lisent les données et construisent une structure de données puis les lignes 8 à 13 analysent les données et impriment un rapport. Nous allons obtenir une table de hachage dont les clés sont les noms de pays et dont les valeurs sont des références vers des tableaux de noms de villes. La structure de données ressemblera à cela :



Commençons par détailler la seconde partie du code. Supposons donc que nous avons déjà cette structure. Comment l'afficher ?

```
 8  foreach $pays (sort keys %table) {
 9    print "$pays: ";
10    my @villes = @{$table{$pays}};
11    print join ', ', sort @villes;
12    print ".\n";
13  }
```