perllol

Table des matières

1 NAME/NOM 1

2 DESCRIPTION 1

2.1 Déclaration et accès aux tableaux de tableaux 1

2.2 Développer la vôtre 2

2.3 Accès et sortie 4

2.4 Tranches 5

3 VOIR AUSSI 5

4 AUTEUR 6

5 TRADUCTION 6

5.1 Version 6

5.2 Traducteur 6

5.3 Relecture 6

6 À propos de ce document 6

1 NAME/NOM

perllol - Manipulation des tableaux de tableaux en Perl

2 DESCRIPTION

2.1 Déclaration et accès aux tableaux de tableaux

La chose la plus simple à construire est un tableau de tableaux (parfois appelé de façon peu précise une liste de listes). C'est raisonnablement facile à comprendre et presque tout ce qui s'y applique pourra aussi être appliqué par la suite aux structures de données plus fantaisistes.

Un tableau de tableau est juste un bon vieux tableau @TdT auquel vous pouvez accéder avec deux indices, comme `$TdT[3][2]`. Voici une déclaration du tableau :

```
    # affecte à notre tableau un tableau de références à des tableaux
    @TdT = (
           [ "fred", "barney" ],
           [ "george", "jane", "elroy" ],
           [ "homer", "marge", "bart" ],
    );

    print $TdT[2][2];
  bart
```

Maintenant, vous devez faire bien attention au fait que les parenthèses extérieures sont bien des parenthèses et pas des accolades ou des crochets. C'est parce que vous affectez dans un @tableau. Vous avez donc besoin de parenthèses. Si vous *n'*aviez *pas* voulu que cela soit un @TdT, mais plutôt une référence à lui, vous auriez pu faire quelque chose du style de ceci :

```
# affecte une référence à une liste de références de liste
$ref_to_TdT = [
    [ "fred", "barney", "pebbles", "bambam", "dino", ],
    [ "homer", "bart", "marge", "maggie", ],
    [ "george", "jane", "elroy", "judy", ],
];

print $ref_to_TdT->[2][2];
```

Notez que le type des parenthèses extérieures a changé et donc notre syntaxe d'accès aussi. C'est parce que, contrairement au C, en Perl vous ne pouvez pas librement échanger les tableaux et leurs références. `$ref_to_TdT` est une référence à un tableau, tandis que `@TdT` est un tableau proprement dit. De la même manière, `$TdT[2]` n'est pas un tableau, mais une référence à un tableau. Ainsi donc vous pouvez écrire ceci :

```
$TdT[2][2]
$ref_to_TdT->[2][2]
```

au lieu de devoir écrire ceci :

```
$TdT[2]->[2]
$ref_to_TdT->[2]->[2]
```

Vous pouvez le faire car la règle dit que, entre des crochets ou des accolades adjacents, vous êtes libre d'omettre la flèche de déréférencement. Mais vous ne pouvez pas faire cela pour la toute première flèche si c'est un scalaire contenant une référence, ce qui signifie que $ref_to_TdT en a toujours besoin.

2.2 Développer la vôtre

Tout ceci est bel et bien pour la déclaration d'une structure de données fixe, mais si vous voulez ajouter de nouveaux éléments à la volée, ou tout contruire à partir de zéro ?

Tout d'abord, étudions sa lecture à partir d'un fichier. C'est quelque chose comme ajouter une rangée à la fois. Nous présumerons qu'il existe un fichier tout simple dans lequel chaque ligne est une rangée et chaque mot un élément. Voici la bonne façon de le faire si vous essayez de développer un tableau @TdT les contenant tous :

```
while (<>) {
    @tmp = split;
    push @TdT, [ @tmp ];
}
```

Vous auriez aussi pu charger tout cela dans une fonction :

```
for $i ( 1 .. 10 ) {
    $TdT[$i] = [ somefunc($i) ];
}
```

Ou vous auriez pu avoir une variable temporaire traînant dans le coin et contenant le tableau.

```
for $i ( 1 .. 10 ) {
    @tmp = somefunc($i);
    $TdT[$i] = [ @tmp ];
}
```

Il est très important que vous vous assuriez d'utiliser le constructeur de référence de tableau `[]`. C'est parce que ceci serait très mauvais :

```
$TdT[$i] = @tmp;
```

Voyez vous, affecter comme ceci un tableau nommé à un scalaire ne fait que compter le nombre d'éléments dans @tmp, ce qui n'est probablement pas ce que vous désirez.

Si vous utilisez `use strict`, vous devrez ajouter quelques déclarations pour que tout fonctionne :

```
use strict;
my(@TdT, @tmp);
while (<>) {
    @tmp = split;
    push @TdT, [ @tmp ];
}
```

Bien sûr, vous n'avez pas du tout besoin de donner un nom au tableau temporaire :

```
while (<>) {
    push @TdT, [ split ];
}
```

Vous n'êtes pas non plus obligé d'utiliser push(). Vous pourriez juste faire une affectation directe si vous savez où vous voulez le mettre :

```
my (@TdT, $i, $line);
for $i ( 0 .. 10 ) {
    $line = <>;
    $TdT[$i] = [ split ' ', $line ];
}
```

ou même juste :

```
my (@TdT, $i);
for $i ( 0 .. 10 ) {
    $TdT[$i] = [ split ' ', <> ];
}
```

Vous devriez en général lorgner d'un regard mauvais l'usage de fonctions qui peuvent potentiellement retourner des listes dans un contexte scalaire sans le formuler explicitement. Ceci sera plus clair pour le lecteur de passage :

```
my (@TdT, $i);
for $i ( 0 .. 10 ) {
    $TdT[$i] = [ split ' ', scalar(<>) ];
}
```

Si vous voulez utiliser une variable $ref_to_TdT comme référence à un tableau, vous devez faire quelque chose comme ceci :

```
while (<>) {
    push @$ref_to_TdT, [ split ];
}
```

Maintenant vous pouvez ajouter de nouvelles rangées. Et pour ajouter de nouvelles colonnes ? Si vous traitez juste des matrices, le plus facile est souvent d'utiliser une simple affectation :

```
for $x (1 .. 10) {
    for $y (1 .. 10) {
        $TdT[$x][$y] = func($x, $y);
    }
}

for $x ( 3, 7, 9 ) {
    $TdT[$x][20] += func2($x);
}
```

Peu importe que ces éléments soient déjà là ou pas : ils seront créés joyeusement pour vous et, si besoin, les éléments intermédiaires seront initialisés à `undef`.
Si vous voulez juste en ajouter à une rangée, vous devrez faire quelque chose ayant l'air un peu plus bizarre :

```
# ajoute de nouvelles colonnes à une rangée existante
push @{ $TdT[0] }, "wilma", "betty";
```

Remarquez qu'on *ne pourrait pas* juste dire :

```
push $TdT[0], "wilma", "betty";  # FAUX !
```

En fait, cela ne se compilerait même pas. Pourquoi donc ? Parce que l'argument de push() doit être un véritable tableau, et non pas une simple référence.

2.3 Accès et sortie

Maintenant il est temps de sortir votre structure de données. Comment allez-vous faire une telle chose ? Eh bien, si vous voulez uniquement l'un des éléments, c'est trivial :

```
print $TdT[0][0];
```

Si vous voulez sortir toute la chose, toutefois, vous ne pouvez pas dire :

```
print @TdT;          # FAUX
```

car vous obtiendrez juste la liste des références et perl ne déréférencera jamais automatiquement les choses pour vous. Au lieu de cela, vous devez faire tourner une boucle ou deux. Ceci imprime toute la structure, en utilisant la construction for() dans le style du shell pour boucler d'un bout à l'autre de l'ensemble des indices extérieurs.

```
for $aref ( @TdT ) {
    print "\t [ @$aref ],\n";
}
```

Si vous voulez garder la trace des indices, vous pouvez faire ceci :

```
for $i ( 0 .. $#TdT ) {
    print "\t elt $i is [ @{$TdT[$i]} ],\n";
}
```

ou peut-être même ceci. Remarquez la boucle intérieure.

```
for $i ( 0 .. $#TdT ) {
    for $j ( 0 .. $#{$TdT[$i]} ) {
        print "elt $i $j is $TdT[$i][$j]\n";
    }
}
```

Comme vous pouvez le voir, cela devient un peu compliqué. C'est pourquoi il est parfois plus facile de prendre une variable temporaire en chemin :

```
for $i ( 0 .. $#TdT ) {
    $aref = $TdT[$i];
    for $j ( 0 .. $#{$aref} ) {
        print "elt $i $j is $TdT[$i][$j]\n";
    }
}
```

Hmm... c'est encore un peu laid. Pourquoi pas ceci :

```
for $i ( 0 .. $#TdT ) {
    $aref = $TdT[$i];
    $n = @$aref - 1;
    for $j ( 0 .. $n ) {
        print "elt $i $j is $TdT[$i][$j]\n";
    }
}
```

2.4 Tranches

Si vous voulez accèder à une tranche (une partie d'une rangée) d'un tableau multidimensionnel, vous allez devoir faire un peu d'indiçage fantaisiste. Car, tandis que nous avons un joli synonyme pour les éléments seuls via la flèche de pointeur pour le déréférencement, il n'existe pas de telle commodité pour les tranches (souvenez-vous, bien sûr, que vous pouvez toujours écrire une boucle pour effectuer une opération sur une tranche).

Voici comment faire une opération en utilisant une boucle. Nous supposerons avoir une variable @TdT comme précédemment.

```
@part = ();
$x = 4;
for ($y = 7; $y < 13; $y++) {
    push @part, $TdT[$x][$y];
}
```

Cette même boucle peut être remplacée par une opération de tranche :

```
@part = @{ $TdT[4] } [ 7..12 ];
```

mais comme vous pouvez l'imaginer, c'est plutôt rude pour le lecteur.

Et si vous vouliez une *tranche à deux dimensions*, telle que $x varie dans 4..8 et $y dans 7 à 12 ? Hmm... voici la façon simple :

```
@newTdT = ();
for ($startx = $x = 4; $x <= 8; $x++) {
    for ($starty = $y = 7; $y <= 12; $y++) {
        $newTdT[$x - $startx][$y - $starty] = $TdT[$x][$y];
    }
}
```

Nous pouvons réduire une partie du bouclage via des tranches.

```
for ($x = 4; $x <= 8; $x++) {
    push @newTdT, [ @{ $LoL[$x] } [ 7..12 ] ];
}
```

Si vous faisiez des transformations schwartziennes, vous auriez probablement choisi map pour cela :

```
@newTdT = map { [ @{ $TdT[$_] } [ 7..12 ] ] } 4 .. 8;
```

Bien sûr, si votre directeur vous accuse de rechercher la sécurité de l'emploi (ou l'insécurité rapide) à l'aide d'un code indéchiffrable, il sera difficile d'argumenter. :-) Si j'étais vous, je mettrais cela dans une fonction :

```
@newTdT = splice_2D( \@LoL, 4 => 8, 7 => 12 );
sub splice_2D {
    my $lrr = shift;     # réf à un tableau de réfs. de tableau !
    my ($x_lo, $x_hi,
        $y_lo, $y_hi) = @_;

    return map {
        [ @{ $lrr->[$_] } [ $y_lo .. $y_hi ] ]
    } $x_lo .. $x_hi;
}
```

3 VOIR AUSSI

perldata, *perlref*, *perldsc*.

4 AUTEUR

Tom Christiansen *<tchrist@perl.com>*

Dernière mise à jour : Thu Jun 4 16:16:23 MDT 1998

5 TRADUCTION

5.1 Version

Cette traduction française correspond à la version anglaise distribuée avec perl 5.10.0. Pour en savoir plus concernant ces traductions, consultez http://perl.enstimac.fr/.

5.2 Traducteur

Roland Trique *<roland.trique@free.fr>*.

5.3 Relecture

Régis Julié *<regis.julie@cetelem.fr>*. Paul Gaborit (Paul.Gaborit at enstimac.fr).

6 À propos de ce document

Ce document est la traduction française du document orginal distribué avec perl. Vous pouvez retrouver l'ensemble de la documentation française Perl (éventuellement mise à jour) en consultant l'URL <http://perl.enstimac.fr/>.

Ce document PDF a été produit Paul Gaborit. Si vous utilisez la version PDF de cette documentation (ou une version papier issue de la version PDF) pour tout autre usage qu'un usage personnel, je vous serai reconnaissant de m'en informer par un petit message <mailto:Paul.Gaborit@enstimac.fr>.

Si vous avez des remarques concernant ce document, en premier lieu, contactez la traducteur (vous devriez trouver son adresse électronique dans la rubrique TRADUCTION) et expliquez-lui gentiment vos remarques ou critiques. Il devrait normalement vous répondre et prendre en compte votre avis. En l'absence de réponse, vous pouvez éventuellement me contacter.

Vous pouvez aussi participer à l'effort de traduction de la documentation Perl. Toutes les bonnes volontés sont les bienvenues. Vous devriez trouver tous les renseignements nécessaires en consultant l'URL ci-dessus.