perlfaq4

Table des matières

1 NAME/NOM

perlfaq4 - Manipulation de données

2 DESCRIPTION

Cette section de la FAQ répond aux questions liées à la manipulation des nombres, des dates, des chaînes de caractères, des tableaux, des tables de hachage, ainsi qu'à divers problèmes relatifs aux données.

3 Données : nombres

3.1 Pourquoi est-ce que j'obtiens des longs nombres décimaux (ex. 19.9499999999999) à la place du nombre que j'attends (ex. 19.95) ?

En interne, un ordinateur représente les flottants (les nombres à virgule) sous forme binaire. Les ordinateurs ne peuvent pas stocker tous les nombres de manière exacte. Certains nombres réels perdent un peu de précision lors de leur traitement. C'est un problème dû à la manière dont les ordinateurs stockent les nombres et concerne donc tous les langages, pas uniquement Perl.

Voir *perlnumber* pour de plus amples informations sur le stockage des nombres et leurs conversions.

Pour limiter le nombre de décimales dans un nombre, vous pouvez utiliser les fonctions printf ou sprintf. Voir perlop (§**??**) pour plus de détails.

```
printf "%.2f", 10/3;

my $number = sprintf "%.2f", 10/3;
```

3.2 Pourquoi int() ne fonctionne pas bien ?

Votre fonction int() fonctionne certainement très bien. Ce sont les nombres qui ne sont pas ce que vous croyez.

Tout d'abord, voyez la question ci-dessus perlfaq4 (§3.1).

Par exemple, ceci :

```
print int(0.6/0.2-2), "\n";
```

affichera 0 et non 1 sur la plupart des ordinateurs, puisque même des nombres aussi simples que 0.6 et 0.2 ne peuvent pas être représentés exactement par des flottants. Ce que vous espériez être 'trois' ci-dessus et en fait quelque chose comme 2.9999999999999995559.

3.3 Pourquoi mon nombre octal n'est-il pas interprété correctement ?

Perl ne comprend les nombres octaux et hexadécimaux en tant que tels que lorsqu'ils sont utilisés comme des valeurs littérales dans le programme. En Perl, les nombres octaux littéraux doivent commencer par "0" et les hexadécimaux par "0x". S'ils sont lus de quelque part et affectés à une variable, aucune conversion automatique n'a lieu. Pour convertir les valeurs, il faut utiliser explicitement oct() ou hex(). oct() interprète à la fois les nombres hexadécimaux ("0x350"), les nombres octaux ("0350" ou même sans le "0" de tête, comme dans "377") et les nombres binaires ("0b1010"), tandis que hex() ne convertit que les hexadécimaux, avec ou sans l'en-tête "0x", comme pour "0x255", "3A", "ff", ou "baffe". Le transformation inverse (depuis décimal vers octal) peut être obtenue via les formats "%o" et "%O" de sprintf().

Ce problème apparaît fréquemment lorsque l'on essaye d'utiliser les fonctions chmod(), mkdir(), umask(), ou sysopen() qui demandent toutes traditionnellement des permissions en octal.

```
chmod(644,  $file); # FAUX
chmod(0644, $file); # correct
```

Notez l'erreur de la première ligne qui spécifie le nombre décimal 644 au lieu du nombre octal voulu 0644. Le problème apparaît mieux avec :

```
printf("%#o",644); # affiche 01204
```

Vous ne vouliez pas dire `chmod(01204, $file)`... Si vous voulez utilisez des valeurs numériques octales comme arguments de chmod() et autres, alors exprimez les en les préfixant d'un zéro et en n'utilisant ensuite que des chiffres de 0 à 7.

3.4 Perl a-t-il une fonction round() ? Et ceil() (majoration) et floor() (minoration) ? Et des fonctions trigonométriques ?

Il faut retenir que int() ne fait que tronquer vers 0. Pour arrondir à un certain nombre de chiffres, sprintf() ou printf() sont d'habitude la voie la plus simple.

```
printf("%.3f", 3.1415926535);       # affiche 3.142
```

Le module POSIX (élément de la distribution standard de Perl) implémente ceil(), floor(), et d'autres fonctions mathématiques et trigonométriques.

```
use POSIX;
$ceil   = ceil(3.5);                        # 4
$floor  = floor(3.5);                       # 3
```

Dans les versions 5.000 à 5.003 de perl, la trigonométrie était faite par le module Math::Complex. À partir de 5.004, le module Math::Trig (élément de la distribution standard de Perl) implémente les fonctions trigonométriques. En interne, il utilise le module Math::Complex, et quelques fonctions peuvent s'échapper de l'axe des réels vers le plan des complexes, comme par exemple le sinus inverse de 2.

Les arrondis dans des applications financières peuvent avoir des conséquences majeures, et la méthode utilisée se doit d'être spécifiée précisément. Dans ces cas, il vaut mieux ne pas avoir confiance dans un quelconque système d'arrondis utilisé par Perl, mais implémenter sa propre fonction d'arrondis telle qu'elle est nécessaire.

Pour comprendre pourquoi, remarquez comment il vous reste un problème lors d'une progression par cinq centièmes :

```
for ($i = 0; $i < 1.01; $i += 0.05) { printf "%.1f ",$i}

0.0 0.1 0.1 0.2 0.2 0.2 0.3 0.3 0.4 0.4 0.5 0.5 0.6 0.7 0.7
0.8 0.8 0.9 0.9 1.0 1.0
```

Perl n'est pas en faute. C'est pareil qu'en C. L'IEEE dit que nous devons faire comme ça. Les nombres en Perl dont la valeur absolue est un entier inférieur à 2**31 (sur les machines 32 bit) fonctionneront globalement comme des entiers mathématiques. Les autres nombres ne sont pas garantis.

3.5 Comment faire des conversions numériques entre différentes bases, entre différentes représentations ?

Comme d'habitude avec Perl, il y a plusieurs moyens de le faire. Ci-dessous, vous trouverez divers exemples d'approches permettant de faire les conversions courantes entre les représentations des nombres. Ce n'est pas une liste exhaustive.

Certains exemples ci-dessous utilisent le module Bit::Vector du CPAN. Les raisons du choix de Bit::Vector plutôt que des fonctions prédéfinies de perl sont qu'il sait traiter des nombres de n'importe quelle longueur, qu'il est optimisé pour certaines opérations et que sa notation semblera familière au moins à certains programmeurs.

Comment faire une conversion hexadécimal vers décimal

En utilisant la conversion interne de perl de la notation 0x :

```
$dec = 0xDEADBEEF;
```

En utilisant la fonction hex :

```
$dec = hex("DEADBEEF");
```

En utilisant pack :

```
$dec = unpack("N", pack("H8", substr("0" x 8 . "DEADBEEF", -8)));
```

En utilisant le module Bit::Vector du CPAN :

```
use Bit::Vector;
$vec = Bit::Vector->new_Hex(32, "DEADBEEF");
$dec = $vec->to_Dec();
```

Comment faire une conversion décimal vers hexadécimal

En utilisant sprintf :

```
$hex = sprintf("%X", 3735928559); # majuscules A-F
$hex = sprintf("%x", 3735928559); # minuscules a-f
```

En utilisant unpack :

```
$hex = unpack("H*", pack("N", 3735928559));
```

En utilisant Bit::Vector :

```
use Bit::Vector;
$vec = Bit::Vector->new_Dec(32, -559038737);
$hex = $vec->to_Hex();
```

En utilisant Bit::Vector qui accepte un nombre de bits impair :

```
use Bit::Vector;
$vec = Bit::Vector->new_Dec(33, 3735928559);
$vec->Resize(32); # suppression des 0 de remplissage
$hex = $vec->to_Hex();
```

Comment faire une conversion octal vers décimal

En utilisant la conversion interne de Perl des nombres préfixés par un zéro :

```
$dec = 033653337357; # notez le préfixe 0 !
```

En utilisant la fonction oct :

```
$dec = oct("33653337357");
```

En utilisant Bit::Vector:

```
use Bit::Vector;
$vec = Bit::Vector->new(32);
$vec->Chunk_List_Store(3, split(//, reverse "33653337357"));
$dec = $vec->to_Dec();
```

Comment faire une conversion décimal vers octal

En utilisant sprintf :

```
$oct = sprintf("%o", 3735928559);
```

En utilisant Bit::Vector:

```
use Bit::Vector;
$vec = Bit::Vector->new_Dec(32, -559038737);
$oct = reverse join('', $vec->Chunk_List_Read(3));
```

Comment faire une conversion binaire vers décimal

Depuis Perl 5.6, vous pouvez écrire les nombres binaires directement via la notation 0b :

```
$number = 0b10110110;
```

En utilisant oct :

```
my $input = "10110110";
$decimal = oct( "0b$input" );
```

En utilisant pack et ord :

```
$decimal = ord(pack('B8', '10110110'));
```

En utilisant pack et unpack pour les grands nombres :

```
$int = unpack("N", pack("B32",
    substr("0" x 32 . "11110101011011011111011101111", -32)));
$dec = sprintf("%d", $int);

# substr() est utilisée pour compléter le nombre par des zéros
# à gauche afin d'obtenir une chaîne de 32 caractères
```

En utilisant Bit::Vector :

```
$vec = Bit::Vector->new_Bin(32, "11011110101011011011111011101111");
$dec = $vec->to_Dec();
```

Comment faire une conversion décimal vers binaire

En utilisant sprintf (perl 5.6 et plus) :

```
$bin = sprintf("%b", 3735928559);
```

En utilisant unpack :

```
$bin = unpack("B*", pack("N", 3735928559));
```

En utilisant Bit::Vector :

```
use Bit::Vector;
$vec = Bit::Vector->new_Dec(32, -559038737);
$bin = $vec->to_Bin();
```

Les autres conversions (par exemple hex -> oct, bin -> hex, etc.) sont laissées en exercices au lecteur.

3.6 Pourquoi & ne fonctionne-t-il pas comme je le veux ?

Le comportement des opérateurs arithmétiques binaires varie selon qu'ils sont utilisés sur des nombres ou des chaînes. Ces opérateurs traitent une chaîne comme une série de bits et travaillent avec (la chaîne `"3"` est le motif de bits `00110011`). Ces opérateurs travaillent avec la forme binaire d'un nombre (le nombre `3` est traité comme le motif de bits `00000011`).

Le fait de dire `11 & 3` effectue donc l'opération "et" bit-à-bit sur des nombres (donnant ici `3`). Le fait de dire `"11" & "3"` effectue un "et" bit-à-bit sur des chaînes (donnant `"1"`).

La plupart des problèmes posés par `&` et | surviennent parce que le programmeur pense qu'il traite un nombre alors qu'en fait c'est une chaîne. Les autres problèmes se posent parce que le programmeur dit :

```
if ("\020\020" & "\101\101") {
  # ...
}
```

mais une chaîne constituée de deux octets nuls (le résultat de `"\020\020" & "\101\101"`) n'est pas une valeur fausse en Perl. Vous avez besoin d'écrire :

```
if ( ("\020\020" & "\101\101") !~ /[^\000]/) {
  # ...
}
```

3.7 Comment multiplier des matrices ?

Utiliser les modules Math::Matrix ou Math::MatrixReal (disponibles sur le CPAN) ou l'extension PDL (également disponible sur le CPAN).

3.8 Comment effectuer une opération sur une série d'entiers ?

Pour appeler une fonction sur chaque élément d'un tableau, et récupérer le résultat, utiliser :

```
@results = map { my_func($_) } @array;
```

Par exemple :

```
@triple = map { 3 * $_ } @single;
```

Pour appeler une fonction sur chaque élément d'un tableau, mais sans tenir compte du résultat :

```
foreach $iterator (@array) {
    some_func($iterator);
}
```

Pour appeler une fonction sur chaque entier d'un (petit) intervalle, on **peut** utiliser :

```
@results = map { some_func($_) } (5 .. 25);
```

mais il faut être conscient que l'opérateur `..` crée un tableau de tous les entiers de l'intervalle utilisant beaucoup de mémoire pour de grands intervalles. Il vaut mieux utiliser :

```
@results = ();
for ($i=5; $i < 500_005; $i++) {
    push(@results, some_func($i));
}
```

Cette situation a été corrigée dans Perl5.005. L'usage de `..` dans une boucle `for` itérera sur l'intervalle, sans créer tout l'intervalle.

```
for my $i (5 .. 500_005) {
    push(@results, some_func($i));
}
```

ne créera pas une liste de 500 000 entiers.

3.9 Comment produire des chiffres romains ?

Récupérez et utilisez le module http://www.cpan.org/modules/by-module/Roman.

3.10 Pourquoi mes nombres aléatoires ne sont-ils pas aléatoires ?

Si vous utilisez une version de Perl antérieure à la 5.004, vous devez appeler `srand` une fois au début de votre programme pour ensemencer le générateur de nombres aléatoires.

```
BEGIN { srand() if $] < 5.004 }
```

Les versions 5.004 et supérieures appellent automatiquement `srand` dès le début. N'appelez pas `srand` plus d'une fois – vous rendriez vos nombres moins aléatoires, plutôt que l'inverse.

Les ordinateurs sont doués pour être déterministes et mauvais lorsqu'il s'agit d'être aléatoires (malgré les apparences provoquées par les bogues dans vos programmes :-). Lisez l'article *random* par Tom Phoenix dans la collection "Far More Than You Ever Wanted To Know" sur http://www.cpan.org/misc/olddoc/FMTEYEWTK.tgz pour en savoir plus à ce sujet. John von Neumann disait : « Quiconque tente de produire des nombres aléatoires par des moyens déterministes vit dans le péché. »

Si vous désirez des nombres qui soient plus aléatoires que ce que fournit `rand` avec `srand`, jetez aussi un oeil au module Math::TrulyRandom sur le CPAN. Il utilise des imperfections de l'horloge du système pour générer des nombres aléatoires, mais ceci prend un certain temps. Si vous voulez un meilleur générateur pseudo-aléatoire que celui qui vient avec le système d'exploitation, lisez les « Numerical Recipes in C » sur http://www.nr.com/.

3.11 Comment obtenir un nombre aléatoire entre X et Y ?

`rand($x)` retourne un nombre tel que `0 <= rand($x) < $x`. Donc perl peut générer un nombre aléatoire entre 0 et la valeur qui sépare *X* et *Y*.
Et donc, pour obtenir un entier entre 10 et 15 inclus, vous pouvez générer un nombre entre 0 et 5 auquel vous ajouterez 10.

```
my $nombre = 10 + int rand( 15-10+1 );
```

Ensuite, on peut transformer ce calcul pour en faire une fonction générique. Elle choisit un nombre aléatoire entre deux entiers donnés :

```
sub random_int_in ($$) {
  my($min, $max) = @_;
  # On suppose que les deux arguments sont eux-mêmes des entiers !
  return $min if $min == $max;
  ($min, $max) = ($max, $min)  if  $min > $max;
  return $min + int rand(1 + $max - $min);
}
```

4 Données : dates

4.1 Comment trouver le jour ou la semaine de l'année ?

La fonction localtime() retourne le jour de l'année. Appeler sans argument, localtime se base sur l'heure (et la date) courante.

```
$jour_de_l_annee = (localtime)[7];
```

Le module POSIX propose le jour ou la semaine de l'année dans les formats disponible :

```
use POSIX qw/strftime/;
my $jour_de_l_annee    = strftime "%j", localtime;
my $semaine_de_l_annee = strftime "%W", localtime;
```

Pour obtenir le jour de l'année d'une date quelconque, utilisez le module Time::Local pour obtenir un temps en secondes depuis l'origine des temps que vous fournirez comme argument à localtime.

```
use POSIX qw/strftime/;
use Time::Local;
my $semaine_de_l_annee = strftime "%W",
       localtime( timelocal( 0, 0, 0, 18, 11, 1987 ) );
```

Le module Date::Calc propose deux fonctions pour calculer cela :

```
use Date::Calc;
my $jour_de_l_annee    = Day_of_Year(  1987, 12, 18 );
my $semaine_de_l_annee = Week_of_Year( 1987, 12, 18 );
```

4.2 Comment trouver le siècle ou le millénaire actuel ?

Utilisez les fonctions simples suivantes :

```
sub trouve_siecle    {
  return int((((localtime(shift || time))[5] + 1999))/100);
}

sub trouve_millenaire {
  return 1+int((((localtime(shift || time))[5] + 1899))/1000);
}
```

Sur certains systèmes, la fonction strftime() du module POSIX a été étendue d'une façon non standard pour utiliser le format `%C`, et ils prétendent que cela représente le "siècle". Ce n'est pas le cas, puisque sur la plupart de ces systèmes, cela ne représente que les deux premiers chiffres de l'année à quatre chiffres, et ne peut ainsi pas être utilisé pour déterminer de façon fiable le siècle ou le millénaire courant.

4.3 Comment comparer deux dates ou en calculer la différence ?

(contribution de brian d foy)

Vous pouvez tout simplement stocker vos dates sous formes de nombres et les soustraire. Mais les choses ne sont pas toujours aussi simples. Si vous souhaitez utiliser des dates mises en forme, les modules Date::Manip, Date::Calc ou DateTime devraient vous aider.

4.4 Comment convertir une chaîne de caractères en secondes depuis l'origine des temps ?

Si cette chaîne est suffisamment régulière pour avoir toujours le même format, on peut la découper et en passer les morceaux à la fonction `timelocal` du module standard Time::Local. Autrement, regarder les modules Date::Calc et Date::Manip disponibles sur le CPAN.

4.5 Comment trouver le jour du calendrier Julien ?

(contribution de brian d foy et de Dave Cross)

Vous pouvez utiliser le module Time::JulianDay disponible sur le CPAN. Assurez-vous tout de même que c'est bien le jour *Julien* que vous voulez. Chacun ayant son idée sur les jours Julien, regardez http://www.hermetic.ch/cal_stud/jdn.htm pour en savoir plus.

Vous pouvez aussi essayer le module DateTime qui sait convertir une date ou une heure en un jour Julien.

```
$ perl -MDateTime -le'print DateTime->today->jd'
2453401.5
```

Ou le jour Julien modifié

```
$ perl -MDateTime -le'print DateTime->today->mjd'
53401
```

Ou même le jour de l'année (que certains croient être le jour Julien)

```
$ perl -MDateTime -le'print DateTime->today->doy'
31
```

4.6 Comment trouver la date d'hier ?

(contribution de brian d foy)

Utilisez l'un des modules Date. Le module `DateTime` fait cela simplement et vous donne le même moment (même heure) mais hier.

```
use DateTime;

my $hier = DateTime->now->subtract( days => 1 );

print "Hier était $hier\n";
```

Vous pouvez aussi choisir le module `Date::Calc` en utilisant sa fonction Today_and_Now.

```
use Date::Calc qw( Today_and_Now Add_Delta_DHMS );

my @hier = Add_Delta_DHMS( Today_and_Now(), -1, 0, 0, 0 );

print "@hier\n";
```

De nombreuses personnes essaient d'utiliser le temps plutôt que le calendrier pour gérer les dates en supposant que tous les jours ont une durée de vingt quatre heures. Or pour de nombreuses personnes, ils existent deux jours par an où ce n'est pas vrai : les jours de changement d'heure (l'heure d'été et l'heure d'hiver). Laissez donc les modules faire ce boulot.

4.7 Perl a-t-il un problème avec l'an 2000 ? Perl est-il compatible an 2000 ?

Réponse courte : Non, Perl n'a pas de problème avec l'an 2000. Oui, Perl est compatible an 2000 (si ceci veut dire quelque chose). Toutefois, les programmeurs que vous avez embauchés pour l'utiliser ne le sont probablement pas.

Réponse longue : la question demande une vraie compréhension du problème. Perl est juste tout aussi compatible an 2000 que votre crayon – ni plus ni moins. Pouvez-vous utilisez votre crayon pour rédiger un mémo non compatible an 2000 ? Bien sûr que oui. Est-ce la faute du crayon ? Bien sûr que non.

Les fonctions de date et d'heure fournies avec Perl (gmtime et localtime) donnent des informations adéquates pour déterminer l'année bien au-delà de l'an 2000 (2038 marque le début des problèmes pour les machines 32-bits). L'année renvoyée par ces fonctions lorsqu'elles sont utilisées dans un contexte de liste est l'année, moins 1900. Pour les années entre 1910 et 1999 ceci est un nombre à deux chiffres *par hasard*. Pour éviter le problème de l'an 2000, ne traitez tout simplement pas l'année comme un nombre à deux chiffres. Elle ne l'est pas.

Quand gmtime() et localtime() sont utilisées dans un contexte scalaire, elles renvoient une chaîne qui contient l'année écrite en entier. Par exemple, `$timestamp = gmtime(1005613200)` fixe $timestamp à la valeur "Tue Nov 13 01:00:00 2001". Ici encore, il n'y a pas de problème de l'an 2000.

Ceci ne veut pas dire que Perl ne peut pas être utilisé pour créer des programmes qui ne respecteront pas l'an 2000. Il peut l'être. Mais un crayon le peut aussi. La faute en revient à l'utilisateur, pas au langage. Au risque d'indisposer la NRA (National Rifle Association) : « Perl ne viole pas l'an 2000, les gens le font ». Pour un exposé plus complet, voir http://www.perl.org/about/y2k.html.

5 Données : chaînes de caractères

5.1 Comment m'assurer de la validité d'une entrée ?

(contribution de brian d foy)

Il existe plusieurs moyens de vérifier que les valeurs sont bien celles que vous attendez ou que vous acceptez. En plus des exemples spécifiques que vous trouverez dans la FAQ perl, vous pouvez aussi jeter un oeil aux modules dont le nom contient "Assert" ou "Validate" ainsi que d'autres modules comme `Regexp::Common`.

Quelques modules savent valider des types particuliers d'entrées. Par exemple `Business::ISBN`, `Business::CreditCard`, `Email::Valid` et `Data::Validate::IP`.

5.2 Comment supprimer les caractères d'échappement d'une chaîne de caractères ?

Cela dépend de que vous entendez par "caractère d'échappement". Les caractères d'échappement d'URL sont traités dans <perlfaq9>. Les caractères de l'interpréteur de commandes avec des barres obliques inversées (\) sont supprimés par :

```
s/\\(.)/$1/g;
```

Ceci ne convertira pas les `"\n"` ou `"\t"` ni aucun autre caractère spécial.

5.3 Comment enlever des paires de caractères successifs ?

(contribution de brian d foy)

Vous pouvez utiliser l'opérateur de substitution pour trouver des couples de caractères identiques et les remplacer pour un seul caractère. Dans cette substitution, nous cherchons un caractère `(.)`. Les parenthèses de mémorisation stockent le caractère reconnu dans la référence `\1` que nous utilisons immédiatement pour reconnaître le même caractère une seconde fois. Nous remplaçons ensuite ce couple de caractères par le caractère `$1` (qui est l'équivalent de `\1` mais dans la chaîne de remplacement).

```
s/(.)\1/$1/g;
```

On peut aussi utiliser l'opérateur de translittération, `tr///`. Dans cet exemple, la liste de recherche est vide mais nous ajoutons le modificateur `c` pour utiliser en fait le complémentaire de cette liste, c'est à dire tout. La liste de remplacement, elle aussi, ne contient rien et donc la translittération est une opération vide puisque nous n'effectuons aucun remplacement (ou plus exactement, nous remplaçons chaque caractère par lui-même). Par contre, comme nous avons ajouter le modificateur `s`, les caractères consécutifs identiques sont remplacés par une seule occurrence.

```
my $str = 'Haarlem'; # aux Pays-bas
$str =~ tr//cs;      # Harlem, comme à New-York
```

5.4 Comment effectuer des appels de fonction dans une chaîne ?

(contribution de brian d foy)

Ceci est documenté dans *perlref* et, bien que ce ne soit pas la chose la plus facile à lire, ça marche. Dans chacun des exemples suivants, nous appelons la fonction à l'intérieur des accolades en utilisant le déréférencement d'une référence. Si nous avons plus d'un seul résultat, nous pouvons construire et déréfencer un tableau anonyme. Dans ce cas, nous appelons la fonction dans un contexte de liste :

```
print "The time values are @{ [localtime] }.\n";
```

Si nous voulons appeler la fonction dans un contexte scalaire, il nous faut faire un petit effort. Il est possible de placer n'importe quel code entre les accolades donc il suffit de placer un code qui retourne une référence à un scalaire :

```
print "La date est ${\(scalar localtime)}.\n"

print "La date est ${ my $x = localtime; \$x }.\n";
```

Si votre fonction retourne déjà une référence, vous n'avez pas besoin de la construire vous-même :

```
sub timestamp { my $t = localtime; \$t }

print "La date est ${ timestamp() }.\n";
```

Le module `Interpolation` peut aussi faire un peu de magie pour vous. Vous pouvez spécifier un nom, dans notre cas `E`, liée à une table de hachage qui fera l'interpolation pour vous. Il y a encore plusieurs autres moyens de faire cela.

```
use Interpolation E => 'eval';
print "The time values are $E{localtime()}.\n";
```

Dans la plupart des cas, il est probablement plus simple d'utiliser la concaténation de chaîne qui impose un contexte scalaire :

```
print "La date est " . localtime . ".\n";
```

5.5 Comment repérer des éléments appariés ou imbriqués ?

Ceci ne peut être réalisé en une seule expression régulière, même très compliquée. Pour trouver quelque chose se situant entre deux caractères simples, un motif comme `/x([^x]*)x/` mettra les morceaux de l'intervalle dans $1. Lorsque le séparateur est de plusieurs caractères, il faudrait en utiliser un ressemblant plus à `/alpha(.*?)omega/`. Mais aucun de des deux ne gère les motifs imbriqués. Pour les expressions imbriquées utilisant `(`, `{`, C([] ou < comme délimiteurs, utilisez le module CPAN Regexp::Common ou voyez "(??{ code })" in *perlre*. Pour les autres cas, il vous faudra écrire un analyseur.

Si vous songez sérieusement à écrire un analyseur syntaxique, de nombreux modules ou gadgets pourront vous rendre la vie plus facile. Il y a les modules du CPAN Parse::RecDescent, Parse::Yapp et Text::Balanced ainsi que le programme byacc. Depuis perl 5.8, Text::Balanced fait partie de la distribution standard.

Une approche simple, mais destructive, consiste à partir de l'intérieur pour aller vers l'extérieur en retirant les plus petites parties imbriquées les unes après les autres :

```
while (s/BEGIN((?:(?!BEGIN)(?!END).)*)END//gs) {
  # faire quelque chose de $1
}
```

Une approche plus complexe et contorsionnée est de faire faire le travail par le moteur d'expressions rationnelles de Perl. Ce qui suit est une courtoisie de Dean Inada, et a plutôt l'allure d'une entrée de l'Obfuscated Perl Contest, mais ce code fonctionne vraiment :

```
  # $_ contient la chaîne à analyser
  # BEGIN et END sont les marqueurs ouvrant et fermants pour le
  # texte inclus.

  @( = ('(','');
  @) = (')','');
  ($re=$_)=~s/((BEGIN)|(END)|.)/$)[!$3]\Q$1\E$([!$2]/gs;
  @$ = (eval{/$re/},$@!~/unmatched/i);
  print join("\n",@$[0..$#$]) if( $$[-1] );
```

5.6 Comment inverser une chaîne de caractères ?

Utiliser reverse() dans un contexte scalaire, tel qu'indiqué dans reverse in *perlfunc*.

```
$reversed = reverse $string;
```

5.7 Comment développer les tabulations dans une chaîne de caractères ?

On peut le faire soi-même :

```
1 while $string =~ s/\t+/' ' x (length($&) * 8 - length($`) % 8)/e;
```

Ou utiliser simplement le module Text::Tabs (qui fait partie de la distribution standard de Perl).

```
use Text::Tabs;
@expanded_lines = expand(@lines_with_tabs);
```

5.8 Comment remettre en forme un paragraphe ?

Utiliser Text::Warp (qui fait partie de la distribution standard de Perl) :

```
use Text::Wrap;
print wrap("\t", '  ', @paragraphs);
```

Les paragraphes passés en argument à Text::Warp ne doivent pas contenir de caractère de saut de ligne. Text::Wrap ne justifie pas les lignes (alignées à droite).

Ou utilisez le module CPAN Text::Autoformat. Les formatage de fichiers s'effectue aisément via un alias shell comme :

```
alias fmt="perl -i -MText::Autoformat -n0777 \
  -e 'print autoformat $_, {all=>1}' $*"
```

Voir la documentation de Text::Autoformat pour connaître toutes ses possibilités.

5.9 Comment accéder à ou modifier N caractères d'une chaîne de caractères ?

Vous pouvez accéder aux premiers caractères d'une chaîne via substr(). Pour obtenir le premier caractère, par exemple, commencer à la position 0 et récupérer une sous-chaîne de longueur 1.

```
$string = "Just another Perl Hacker";
$first_char = substr( $string, 0, 1 );  #  'J'
```

Pour modifier une partie d'une chaîne, vous pouvez utiliser le quatrième argument optionnel qui est la chaîne de remplacement.

```
substr( $string, 13, 4, "Perl 5.8.0" );
```

Vous pouvez aussi utiliser substr() en tant que lvalue :

```
substr($a, 0, 3) = "Tom";
```

5.10 Comment changer la nième occurrence de quelque chose ?

Il faut conserver soi-même l'évolution de n. Par exemple, si l'on souhaite changer la cinquième occurrence de "whoever" ou "whomever" en "whosoever" ou "whomsoever", sans tenir compte de la casse. Supposons dans la suite que $_ contienne la chaîne à modifier.

```
$count = 0;
s{((whom?)ever)}{
    ++$count == 5          # Est-ce la cinquième ?
       ? "${2}soever"      # oui: faire l'échange
       : $1                # non: le laisser tel quel
}ige;
```

Dans des cas plus généraux, on peut utiliser le modificateur /g dans une boucle while, en comptant le nombre de correspondances.

```
$WANT = 3;
$count = 0;
$_ = "One fish two fish red fish blue fish";
while (/(\w+)\s+fish\b/gi) {
    if (++$count == $WANT) {
        print "The third fish is a $1 one.\n";
    }
}
```

Ceci sort: "The third fish is a red one." On peut aussi utiliser un compteur de répétition, et un motif répété comme ceci :

```
/(?:\w+\s+fish\s+){2}(\w+)\s+fish/i;
```

5.11 Comment compter le nombre d'occurrences d'une sous-chaîne dans une chaîne de caractères ?

Plusieurs moyens sont possibles, avec une efficacité variable. Pour trouver le nombre d'un caractère particulier (X) dans une chaîne, on peut utiliser la fonction tr/// comme ceci :

```
$string = "ThisXlineXhasXsomeXx'sXinXit";
$count = ($string =~ tr/X//);
print "There are $count X characters in the string";
```

Ceci marche bien lorsque l'on cherche un seul caractère. Cependant si l'on essaye de compter des sous-chaînes de plusieurs caractères à l'intérieur d'une chaîne plus grande, tr/// ne marchera pas. On peut alors envelopper d'une boucle while() une recherche de motif globale. Par exemple, comptons les entiers négatifs :

```
$string = "-9 55 48 -2 23 -76 4 14 -44";
while ($string =~ /-\d+/g) { $count++ }
print "There are $count negative numbers in the string";
```

Une autre manière de faire utilise la reconnaissance globale dans un contexte de liste puis assigne le résultat à un scalaire, produisant ainsi le nombre de reconnaissances.

```
$count = () = $string =~ /-\d+/g;
```

5.12 Comment mettre en majuscule toutes les premières lettre des mots d'une ligne ?

Pour mettre en lettres majuscules toutes les premières lettres des mots :

```
    $line =~ s/\b(\w)/\U$1/g;
```

Ceci a pour effet de bord de transformer "aujourd'hui" en "Aujourd'Hui". C'est parfois ce qu'on veut. Sinon, on peut préférer cette solution (suggérée par brian d foy) :

```
$string =~ s/ (
              (^\w)    #au début d'une ligne
                |      # ou
              (\s\w)   #précédé d'un espace
                )
             /\U$1/xg;
$string =~ /([\w']+)/\u\L$1/g;
```

Pour transformer toute la ligne en lettres majuscules :

```
    $line = uc($line);
```

À l'inverse, pour avoir chaque mot en lettres minuscules, avec la première lettre majuscule :

```
    $line =~ s/(\w+)/\u\L$1/g;
```

On peut (et l'on devrait toujours) rendre ces caractères sensibles aux locales en plaçant une directive `use locale` dans le programme. Voir *perllocale* pour d'interminables détails sur les locales.

On s'y réfère parfois comme consistant à mettre quelque chose en "casse de titre", mais ce n'est pas tout à fait juste. Observez la capitalisation correcte du film *Dr. Strangelove or : How I Learned to Stop Worrying and Love the Bomb*, par exemple. (NdT : d'autant que l'usage en français est de ne mettre de majuscule qu'en début de phrase et aux noms propres.)

Le module Text::Autoformat de Domian Conway fournit quelques transformations sympathiques :

```
use Text::Autoformat;
my $x = "Dr. Strangelove or: How I Learned to Stop ".
   "Worrying and Love the Bomb";

print $x, "\n";
for my $style (qw( sentence title highlight ))
  {
    print autoformat($x, { case => $style }), "\n";
  }
```

5.13 Comment découper une chaîne séparée par un [caractère] sauf à l'intérieur d'un [caractère] ?

Plusieurs modules savent gérer ce genre d'analyses – Text::Balanced, Text::CSV, Text::CSV_XS et Text::ParseWord parmi d'autres.

Prenons l'exemple du cas où l'on souhaite découper une chaîne qui est subdivisée en différents champs par des virgules. On ne peut utiliser `split(/,/)` parce qu'il ne faudrait pas découper si la virgule est entre guillemets. Par exemple pour la ligne de donnée suivante :

```
    SAR001,"","Cimetrix, Inc","Bob Smith","CAM",N,8,1,0,7,"Error, Core Dumped"
```

À cause de la restriction imposée par les guillemets, ceci devient un problème relativement complexe. Heureusement, nous avons Jeffrey Friedl, auteur du livre *Mastering Regular Expressions*, qui s'est penché sur le problème pour nous. Il suggère (en supposant que $text contient la chaîne) :

```
@new = ();
push(@new, $+) while $text =~ m{
    "([^\"\\]*(?:\\.[^\"\\]*)*)",?  # regroupe les éléments entre guillemets
  | ([^,]+),?
  | ,
}gx;
push(@new, undef) if substr($text,-1,1) eq ',';
```

Pour représenter un vrai guillemet à l'intérieur d'un champ délimité par des guillemets, il faut le protéger d'une barre oblique inverse comme caractère d'échappement (c.-à-d. `"comme \"ceci\""`).

Comme autre choix, le module Text::ParseWords (qui fait partie de la distribution standard de Perl) permet de faire :

```
use Text::ParseWords;
@new = quotewords(",", 0, $text);
```

Il existe aussi un module Text::CSV (Comma-Separated Values – Valeurs séparées par des virgules) sur le CPAN.

5.14 Comment supprimer des espaces blancs au début/à la fin d'une chaîne ?

(contribution de brian d foy)

Une substitution peut le faire pour vous. Pour une simple ligne, vous voulez remplacer tous les espaces en début et en fin de ligne par rien du tout. Cette double substitution le fait :

```
s/^\s+//;
s/\s+$//;
```

Vous pourriez aussi l'écrire en une seule substitution bien que ce soit plus lent. Mais ce n'est peut-être pas important pour vous.

```
s/^\s+|\s+$//g;
```

Dans cette expression rationnelle, l'alternative est reconnue soit en début soit en fin de chaîne puisque les ancres ont une précédence plus faible que l'alternative. Avec le modificateur `/g`, la substitution s'effectue partout où elle est reconnue, et donc au début et à la fin. Souvenez-vous que `\s+` reconnaît les caractères de fin de ligne et que `$` peut s'ancrer à la fin de la chaîne, donc le caractère de fin de ligne disparaîtra lui-aussi. Ajoutez juste le saut de ligne en sortie, ce qui a l'avantage de conserver les lignes entièrement blanches qui sont vidées par `^\s+`.

```
while( <> )
  {
    s/^\s+|\s+$//g;
    print "$_\n";
  }
```

Pour une chaîne multi-lignes, vous pouvez appliquer l'expression rationnelle à chaque ligne logique de la chaîne en ajoutant le modificateur `/m` (pour "multi-lignes"). Avec ce modificateur, `$` est reconnu *avant* un saut de ligne, et donc il n'est pas supprimé. Par contre, il supprimera encore le saut de ligne en fin de chaîne.

```
$string =~ s/^\s+|\s+$//gm;
```

Souvenez-vous que les lignes entièrement blanches vont disparaître puisque la première partie de l'alternative les remplacera par rien. Si vous souhaitez les lignes blanches, il faut faire un peu plus de travail. Au lieu de reconnaître n'importe quel espace (puisque cela inclut les sauts de ligne), ne reconnaissons que les autres espaces.

```
$string =~ s/^[\t\f ]+|[\t\f ]+$//mg;
```

5.15 Comment cadrer une chaîne avec des blancs ou un nombre avec des zéros ?

(Cette réponse est une contribution de Uri Guttman, avec un coup de main de Bart Lateur).

Dans les exemples suivants, `$pad_len` est la longueur dans laquelle vous voulez cadrer la chaîne, `$text` ou `$num` contient la chaîne à cadrer, et `$pad_char` contient le caractère de remplissage. Vous pouvez utiliser une constante contenant une chaîne d'un seul caractère à la place de la variable `$pad_char` si vous savez ce que c'est. Et de la même façon vous pouvez utiliser un entier à la place de `$pad_len` si vous connaissez à l'avance la longueur du cadrage.

La méthode la plus simple utilise la fonction `sprintf`. Elle peut cadrer à gauche et à droite avec des espaces et à gauche avec des zéros et elle ne tronquera pas le résultat. La fonction `pack` peut seulement cadrer des chaînes à droite avec des blancs et elle tronquera le résultat à la longueur maximale de `$pad_len`.

```
# Cadrage à gauche d'une chaîne avec des blancs (pas de troncature)
$padded = sprintf("%${pad_len}s", $text);
$padded = sprintf("%*s", $pad_len, $text);  # idem

# Cadrage à droite d'une chaîne avec des blancs (pas de troncature)
$padded = sprintf("%-${pad_len}s", $text);
$padded = sprintf("%-*s", $pad_len, $text); # idem

# Cadrage à gauche d'un nombre avec 0 (pas de troncature)
$padded = sprintf("%0${pad_len}d", $num);
$padded = sprintf("%0*d", $pad_len, $num); # idem

# Cadrage à droite d'un nombre avec 0 (avec troncature)
$padded = pack("A$pad_len",$text);
```

Si vous avez besoin de cadrer avec un caractère autre qu'un blanc ou un zéro, vous pouvez utiliser une des méthodes suivantes. Elles génèrent toutes une chaîne de cadrage avec l'opérateur `x` et la combinent avec `$text`. Ces méthodes ne tronquent pas `$text`.

Cadrage à gauche et à droite avec n'importe quel caractère, créant une nouvelle chaîne :

```
$padded = $pad_char x ( $pad_len - length( $text ) ) . $text;
$padded = $text . $pad_char x ( $pad_len - length( $text ) );
```

Cadrage à gauche et à droite avec n'importe quel caractère, modifiant directement `$text` :

```
substr( $text, 0, 0 ) = $pad_char x ( $pad_len - length( $text ) );
$text .= $pad_char x ( $pad_len - length( $text ) );
```

5.16 Comment extraire une sélection de colonnes d'une chaîne de caractères ?

Utiliser les fonctions substr() ou unpack(), toutes deux documentées dans *perlfunc*. Ceux qui préfèrent penser en termes de colonnes plutôt qu'en longueurs de lignes peuvent utiliser ce genre de choses :

```
# déterminer le format de unpack nécessaire pour découper la
# sortie d'un ps de Linux
# les arguments sont les colonnes sur lesquelles découper
my $fmt = cut2fmt(8, 14, 20, 26, 30, 34, 41, 47, 59, 63, 67, 72);
sub cut2fmt {
    my(@positions) = @_;
    my $template  = '';
    my $lastpos   = 1;
    for my $place (@positions) {
        $template .= "A" . ($place - $lastpos) . " ";
        $lastpos   = $place;
    }
    $template .= "A*";
    return $template;
}
```

5.17 Comment calculer la valeur soundex d'une chaîne ?

(contribution de brian d foy)
Vous pouvez utiliser le module standard Test::Soundex. Si vous souhaitez faire de la reconnaissance floue et approchée, vous devriez aussi essayer les modules String::Approx, Text::Metaphone et Text::DoubleMetaphone.

5.18 Comment interpoler des variables dans des chaînes de texte ?

Supposons une chaîne comme celle-ci ($bar et $foo ne sont pas interpolées) :

```
$text = 'this has a $foo in it and a $bar';
```

Vous pouvez utiliser une substitution avec une double évaluation. Le premier /e remplace `$1` par `$foo` et le second /e remplace `$foo` par sa valeur. Vous pouvez habiller cela par un `eval {}` : si vous essayer d'obtenir la valeur d'une variable non déclaré et si `use strict` est actif, vous aurez une erreur fatale.

```
eval { $text =~ s/(\$\w+)/$1/eeg };
die if $@;
```

Il serait probablement préférable dans le cas général de traiter ces variables comme des entrées dans une table de hachage à part. Par exemple :

```
%user_defs = (
    foo  => 23,
    bar  => 19,
);
$text =~ s/\$(\w+)/$user_defs{$1}/g;
```

5.19 En quoi est-ce un problème de toujours placer "$vars" entre guillemets ?

Le problème est que ces guillemets forcent la conversion en chaîne, imposant aux nombres et aux références de devenir des chaînes, même lorsqu'on ne le souhaite pas. Pensez-y de cette façon : l'expansion des guillemets est utilisée pour produire de nouvelles chaînes. Si vous avez déjà une chaîne, pourquoi en vouloir plus ?
Si l'on prend l'habitude d'écrire des choses bizarres comme ça :

```
print "$var";       # MAL
$new = "$old";      # MAL
somefunc("$var");   # MAL
```

on se retrouve vite avec des problèmes. Les constructions suivantes devraient (dans 99.8% des cas) être plus simples et plus directes :

```
print $var;
$new = $old;
somefunc($var);
```

Autrement, en plus de ralentir, ceci risque de casser le code lorsque ce qui est dans la variable scalaire n'est effectivement ni une chaîne de caractères, ni un nombre mais une référence :

```
func(\@array);
sub func {
    my $aref = shift;
    my $oref = "$aref";  # FAUX
}
```

On peut aussi se retrouver face à des problèmes subtils pour les quelques opérations pour lesquels Perl fait effectivement la différence entre une chaîne de caractères et un nombre, comme pour l'opérateur magique d'autoincrémentation ++, ou pour la fonction syscall().
La mise en chaîne détruit aussi les tableaux :

```
@lines = `command`;
print "@lines";            # FAUX - ajoute des blancs
print @lines;              # correct
```

5.20 Pourquoi est-ce que mes documents <<HERE ne marchent pas ?

Vérifier les trois points suivants :

Il ne doit pas y avoir d'espace après le <<.

Il devrait (habituellement) y avoir un point-virgule à la fin.

On ne peut pas mettre (facilement) d'espace devant la marque.

Si l'on souhaite indenter le texte du document inséré, on peut faire ceci :

```
# tout à la fois
($VAR = <<HERE_TARGET) =~ s/^\s+//gm;
    your text
    goes here
HERE_TARGET
```

Mais la cible HERE_TARGET doit quand même être alignée sur la marge. Pour l'indenter également, il faut mettre l'indentation entre apostrophes.

```
($quote = <<'    FINIS') =~ s/^\s+//gm;
        ...we will have peace, when you and all your works have
        perished--and the works of your dark master to whom you
        would deliver us. You are a liar, Saruman, and a corrupter
        of men's hearts.  --Theoden in /usr/src/perl/taint.c
    FINIS
$quote =~ s/\s+--/\n--/;
```

Une jolie fonction d'usage général pour réarranger proprement des documents insérés indentés est donnée ci-dessous. Elle attend un document inséré en argument. Elle regarde si chaque ligne commence avec une sous-chaîne commune, et si tel est le cas, elle enlève cette sous-chaîne. Dans le cas contraire, elle prend le nombre d'espaces en tête de la première ligne et en enlève autant à chacune des lignes suivantes.

```
sub fix {
    local $_ = shift;
    my ($white, $leader);  # espaces en commun et chaîne en commun
    if (/^\s*(?:([^\w\s]+)(\s*).*\n)(?:\s*\1\2?.*\n)+$/) {
        ($white, $leader) = ($2, quotemeta($1));
    } else {
        ($white, $leader) = (/^(\s+)/, '');
    }
    s/^\s*?$leader(?:$white)?//gm;
    return $_;
}
```

Ceci fonctionne avec des chaînes d'en-tête spéciales et déterminées dynamiquement :

```
$remember_the_main = fix<<'    MAIN_INTERPRETER_LOOP';
    @@@ int
    @@@ runops() {
    @@@     SAVEI32(runlevel);
    @@@     runlevel++;
    @@@     while ( op = (*op->op_ppaddr)() );
    @@@     TAINT_NOT;
    @@@     return 0;
    @@@ }
MAIN_INTERPRETER_LOOP
```

Ou encore avec une quantité fixée d'en-tête d'espaces, tout en conservant correctement l'indentation :

```
$poem = fix<<EVER_ON_AND_ON;
   Now far ahead the Road has gone,
      And I must follow, if I can,
   Pursuing it with eager feet,
      Until it joins some larger way
   Where many paths and errands meet.
      And whither then? I cannot say.
            --Bilbo in /usr/src/perl/pp_ctl.c
EVER_ON_AND_ON
```

6 Données : tableaux

6.1 Quelle est la différence entre une liste et un tableau ?

Un tableau a une longueur variable. Une liste n'en a pas. Un tableau est quelque chose sur laquelle vous pouvez pousser ou retirer des données, alors qu'une liste est un ensemble de valeurs. Certaines personnes font la distinction qu'une liste est une valeur tandis qu'un tableau est une variable. Les sous-programmes se voient passer et renvoient des listes, vous mettez les choses dans un contexte de liste, vous initialisez des tableaux avec des listes, et vous balayez une liste avec foreach(). Les variables `@` sont des tableaux, les tableaux anonymes sont des tableaux, les tableaux dans un contexte scalaire se comportent comme le nombre de leurs éléments, les sous-programmes accèdent à leurs arguments via le tableau `@_` et les fonctions push/pop/shift ne fonctionnent que sur les tableaux.

Une note au passage : il n'existe pas de liste dans un contexte scalaire. Lorsque vous dites

```
$scalar = (2, 5, 7, 9);
```

vous utilisez l'opérateur virgule dans un contexte scalaire, c'est donc l'opérateur virgule scalaire qui est utilisé. Il n'y a jamais eu là de liste du tout ! C'est donc la dernière valeur qui est renvoyée : 9.

6.2 Quelle est la différence entre $array[1] et @array[1] ?

La première est une valeur scalaire tandis que la seconde est une tranche de tableaux, ce qui en fait une liste avec une seule valeur (scalaire). On est censé utiliser $ lorsque l'on désire une valeur scalaire (le plus souvent) et @ lorsque l'on souhaite une liste avec une seule valeur scalaire à l'intérieur (très, très rarement, presque jamais en fait).

Il n'y a souvent pas de différence, mais il arrive que si. Par exemple, comparer :

```
$good[0] = `une commande qui produit plusieurs lignes`;
```

avec

```
@bad[0]  = `une commande qui produit plusieurs lignes`;
```

La directive `use warnings` ou le drapeau **-w** préviennent lorsque de tels problèmes se présentent.

6.3 Comment supprimer les doublons d'une liste ou d'un tableau ?

(contribution de brian d foy)

Utilisez une table de hachage. Lorsque vous pensez à "unique" ou à "doublons", pensez aux "clés de hachage".

Si l'ordre des éléments ne compte pas, vous pouvez créer une table de hachage dont vous extrairez les clés. La manière de créer la table de hachage importe peu : il vous suffit d'utiliser `keys` pour récupérer les éléments uniques.

```
my %hachage   = map { $_, 1 } @tableau;
# ou via une tranche de hachage : @hachage{ @tableau } = ();
# ou via un foreach : $hachage{$_} = 1 foreach ( @tableau );

my @unique = keys %hachage;
```

Vous pouvez aussi parcourir tous les éléments et ne conservez que ceux que vous n'avez encore jamais vus. La première fois que la boucle traite un élément, cet élément n'a pas de clé correspondante dans `%dejavu`. Le condition dans l'instruction commençant par `next` crée cette clé et récupère immédiatement sa valeur (qui est `undef`) avant de l'incrémenter. Comme cette valeur initiale est fausse, le `next` n'a pas lieu et la boucle continue en exécutant le `push`. Lorsque la boucle rencontre à nouveau le même élément, la clé correspondante existe dans la tableau de hachage *et* cette clé est vraie (puisque sa valeur incrémentée n'est plus undef), et donc le next interrompt cette itération et la boucle passe à l'élément suivant.

```
my @unique = ();
my %dejavu = ();

foreach my $elem ( @tableau )
  {
    next if $dejavu{ $elem }++;
    push @unique, $elem;
  }
```

Vous pouvez écrire cela de manière plus concise en utilisant grep, qui fera la même chose.

```
my %dejavu = ();
my @unique = grep { ! $dejavu{ $_ }++ } @tableau;
```

6.4 Comment savoir si une liste ou un tableau inclut un certain élément ?

(cette réponse est, en partie, une contribution de Anno Siegel)

D'entendre le mot "*in*clut" est une *in*dication qu'on aurait dû utiliser un tableau de hachage, et pas une liste ou un tableau, pour enregistrer ses données. Les hachages sont conçus pour répondre rapidement et efficacement à cette question, alors que les tableaux ne le sont pas.

Cela dit il y a plusieurs moyens pour résoudre ce problème. Si vous faites cette requête plusieurs fois sur des valeurs de chaînes arbitraires, le plus rapide est probablement d'inverser le tableau original et de maintenir à jour une table de hachage dont les clefs sont les valeurs du tableau.

```
@blues = qw/azure cerulean teal turquoise lapis-lazuli/;
%is_blue = ();
for (@blues) { $is_blue{$_} = 1 }
```

Vous pouvez alors regarder si telle couleur ($some_color) est du bleu en testant $is_blue{$some_color}. Il aurait été une bonne idée de conserver les bleus dans un hachage dès le début.

Si les valeurs sont de petits entiers positifs, on peut simplement utiliser un tableau indexé. Ce genre de tableau prendra moins de place :

```
@primes = (2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31);
undef @is_tiny_prime;
for (@primes) { $is_tiny_prime[$_] = 1 }
# ou simplement @istiny_prime[@primes] = (1) x @primes;
```

À partir de là pour vérifier si $some_number est un nombre premier il suffit de tester $is_tiny_prime[$some_number].

Si les valeurs en question sont des entiers plutôt que des chaînes de caractères, on économise un certain espace en utilisant des chaînes de bits à la place :

```
@articles = ( 1..10, 150..2000, 2017 );
undef $read;
for (@articles) { vec($read,$_,1) = 1 }
```

Et là vérifier si `vec($read,$n,1)` est vrai pour un `$n`.

Ces méthodes garantissent des test individuels rapides mais nécessitent une réorganisation de la liste ou du tableau original. Cela ne vaut le coup que si vous devez tester plusieurs valeurs du même tableau.

Si vous ne devez faire ce test qu'une seule fois, le module standard List::Util exporte la fonction <first>. Elle s'arrête dès qu'elle trouve l'élément recherché. Elle est écrite en C pour des raisons de performance et son équivalent en Perl ressemblerait au sous-programme suivant :

```
sub first (&@) {
  my $code = shift;
  foreach (@_) {
    return $_ if &{$code}();
  }
  undef;
}
```

Si la performance n'est pas un problème, une manière de faire consiste à appliquer à toutes la liste un grep dans un contexte scalaire (il retourne alors le nombre d'items ayant passé la condition). En revanche, vous y gagnez la possibilité de savoir combien de fois l'élément est reconnu.

```
my $nb_items = grep $_ eq $element, @tableau;
```

Si vous souhaitez récupérer les éléments reconnus, utilisez grep dans un contexte de liste.

```
my @reconnus = grep $_ eq $element, @tableau;
```

6.5 Comment calculer la différence entre deux tableaux ? Comment calculer l'intersection entre deux tableaux ?

Utiliser un hachage. Voici un code pour faire les deux et même plus. Il suppose chaque élément dans un tableau donné :

```
@union = @intersection = @difference = ();
%count = ();
foreach $element (@array1, @array2) { $count{$element}++ }
foreach $element (keys %count) {
    push @union, $element;
    push @{ $count{$element} > 1 ? \@intersection : \@difference }, $element;
}
```

Notez que ceci est la *différence symétrique*, c'est-à-dire tous les éléments qui sont soit dans A, soit dans B, mais pas dans les deux. Voyez cela comme une opération xor (un ou exclusif).

6.6 Comment tester si deux tableaux ou hachages sont égaux ?

Le code suivant fonctionne pour les tableaux à un seul niveau. Il utilise une comparaison de chaînes, et ne distingue pas les chaînes vides définies ou indéfinies. Modifiez-le si vous avez d'autres besoins.

```
$are_equal = compare_arrays(\@frogs, \@toads);

sub compare_arrays {
    my ($first, $second) = @_;
    no warnings;  # silence spurious -w undef complaints
    return 0 unless @$first == @$second;
    for (my $i = 0; $i < @$first; $i++) {
        return 0 if $first->[$i] ne $second->[$i];
    }
    return 1;
}
```

Pour les structures à niveau multiples, vous pouvez désirer utiliser une approche ressemblant plus à celle-ci. Elle utilise le module CPAN FreezeThaw :

```
use FreezeThaw qw(cmpStr);
@a = @b = ( "this", "that", [ "more", "stuff" ] );
```

```
printf "a and b contain %s arrays\n",
    cmpStr(\@a, \@b) == 0
       ? "the same"
       : "different";
```

Cette approche fonctionne aussi pour comparer des hachages. Ici, nous allons démontrer deux réponses différentes :

```
use FreezeThaw qw(cmpStr cmpStrHard);

%a = %b = ( "this" => "that", "extra" => [ "more", "stuff" ] );
$a{EXTRA} = \%b;
$b{EXTRA} = \%a;

printf "a and b contain %s hashes\n",
    cmpStr(\%a, \%b) == 0 ? "the same" : "different";

printf "a and b contain %s hashes\n",
    cmpStrHard(\%a, \%b) == 0 ? "the same" : "different";
```

La première rapporte que les deux hachages contiennent les mêmes données, tandis que la seconde rapporte le contraire. Le fait de déterminer celle que vous préférez vous est laissée en exercice.

6.7 Comment trouver le premier élément d'un tableau vérifiant une condition donnée ?

Pour trouver le premier élément d'un tableau vérifiant une condition, vous pouvez utiliser la fonction `first` du module List::Util (en standard depuis Perl 5.8). L'exemple suivant trouve le premier élément contenant "Perl".

```
use List::Util qw(first);

my $element = first { /Perl/ } @tableau;
```

Si vous ne pouvez pas utiliser List::Util, vous pouvez écrire votre propre boucle pour faire la même chose. Dès que vous trouvez l'élément voulu, vous stoppez la boucle via last().

```
my $found;
foreach ( @tableau )
  {
    if( /Perl/ ) { $found = $_; last }
  }
```

Si vous souhaitez connaître l'indice de l'élément, vous pouvez faire une boucle sur tous les indices du tableau et tester l'élément lié à cet indice jusqu'à trouver l'élément satisfaisant la condition.

```
my( $found, $index ) = ( undef, -1 );
for( $i = 0; $i < @tableau; $i++ )
  {
    if( tableau[$i] =~ /Perl/ )
      {
         $found = tableau[$i];
         $index = $i;
         last;
      }
  }
```

6.8 Comment gérer des listes chaînées ?

En général, avec Perl, on n'a pas de besoin de liste chaînée, puisqu'avec des tableaux usuels, on peut ajouter (push/unshift) et enlever (pop/shift) de part et d'autre, ou l'on peut utiliser splice pour ajouter et/ou enlever un nombre arbitraire d'éléments à un point arbitraire. Pop et shift sont toutes deux des opérations en O(1) sur les tableaux dynamiques de Perl. En absence de shifts et pops, push a en général besoin de faire des réallocations environ toutes les log(N) fois, et unshift aura besoin de copier des pointeurs à chaque fois.

Si vous le voulez vraiment, vous pouvez utiliser les structures décrites dans *perldsc* ou *perltoot* et faire exactement comme le livre d'algorithmes dit de faire. Par exemple, imaginez un noeud de liste tel que celui-ci :

```
$node = {
    VALUE => 42,
    LINK  => undef,
};
```

Vous pouvez balayer la liste de cette façon :

```
print "List: ";
for ($node = $head;  $node; $node = $node->{LINK}) {
    print $node->{VALUE}, " ";
}
print "\n";
```

Vous pouvez allonger la liste ainsi :

```
my ($head, $tail);
$tail = append($head, 1);       # ajoute un élément en tête
for $value ( 2 .. 10 ) {
    $tail = append($tail, $value);
}

sub append {
    my($list, $value) = @_;
    my $node = { VALUE => $value };
    if ($list) {
        $node->{LINK} = $list->{LINK};
        $list->{LINK} = $node;
    } else {
        $_[0] = $node;   # remplace la valeur fournie à l'appel
    }
    return $node;
}
```

Mais encore une fois, les fonctions intégrées de Perl sont presque toujours suffisantes.

6.9 Comment gérer des listes circulaires ?

Des listes circulaires peuvent être gérées de façon traditionnelle par des listes chaînées, ou encore en utilisant simplement quelque chose comme ceci avec un tableau :

```
unshift(@array, pop(@array));  # les derniers seront les premiers
push(@array, shift(@array));   # et vice versa
```

6.10 Comment mélanger le contenu d'un tableau ?

Si vous disposez au moins de la version 5.8.0 de Perl ou si la version 1.03 ou plus de Scalar-List-Utils est installée, vous pouvez faire :

```
use List::Util 'shuffle';

@melange = shuffle(@tableau);
```

Sinon, utilisez le mélangeur de Fisher-Yates :

```
sub melange_de_fisher_yates {
  my $deck = shift;  # $deck est une référence à un tableau
  my $i = @$deck;
  while (--$i) {
    my $j = int rand ($i+1);
    @$deck[$i,$j] = @$deck[$j,$i];
  }
}

# mélange ma collection de mpeg
#
my @mpeg = <audio/*/*.mp3>;
melange_de_fisher_yates( \@mpeg );    # mélange @mpeg sur place
print @mpeg;
```

Notez que le mélangeur ci-dessus mélange le tableau sur place alors que List::Util::shuffle() prend une liste et retourne une nouvelle liste mélangée.

Vous avez probablement vu des algorithmes de mélange qui fonctionnent en utilisant splice, choisissant au hasard un élément à mettre dans l'élément courant.

```
srand;
@new = ();
@old = 1 .. 10;  # just a demo
while (@old) {
    push(@new, splice(@old, rand @old, 1));
}
```

Ceci est mauvais car splice est déjà en O(N), et puisqu'on l'exécute N fois, on vient d'inventer un algorithme quadratique ; c'est-à-dire en O(N**2). Ceci ne supporte pas la montée en charge, même si Perl est tellement efficace qu'on ne le remarquerait probablement pas avant d'atteindre des tableaux relativement grands.

6.11 Comment traiter/modifier chaque élément d'un tableau ?

Utiliser `for`/`foreach` :

```
for (@lines) {
    s/foo/bar/;     # changer ce mot
    tr/XZ/ZX/;       # échanger ces lettres
}
```

En voici un autre ; calculons des volumes sphériques :

```
for (@volumes = @radii) {   # @volumes contient les parties modifiées
    $_ **= 3;
    $_ *= (4/3) * 3.14159;  # ceci sera transformé en une constante
}
```

Ceci pourrait aussi être fait en utilisant map() qui permet de transformer une liste en une autre liste :

```
    @volumes = map {$_ ** 3 * (4/3) * 3.14159} @radii;
```

Si vous souhaitez faire la même chose pour modifier les valeurs d'une table de hachage, vous pouvez utiliser la fonction `values`. Depuis perl 5.6, les valeurs ne sont pas copiées et donc, en modifiant $orbit (dans notre cas), vous modifiez réellement la valeur.

```
for $orbit ( values %orbits ) {
  ($orbit **= 3) *= (4/3) * 3.14159;
}
```

Dans les versions antérieures à perl 5.6, la fonction `values` retournaient des copies des valeurs et donc certains anciens codes peuvent encore contenir la construction `@orbits{keys %orbits}` à la place de `values %orbits`.

6.12 Comment sélectionner aléatoirement un élément d'un tableau ?

Utiliser la fonction rand() (voir rand in *perlfunc*) :

```
$index   = rand @tableau;
$element = $tableau[$index];
```

Ou plus simplement :

```
my $element = $tableau[ rand @tableau ];
```

6.13 Comment générer toutes les permutations des N éléments d'une liste ?

Utilisez le module List::Permutor de CPAN. Si la liste est un véritable tableau, essayez le module Algorithm::Permute (disponible lui aussi dans CPAN). Il est écrit en code XS et est très efficace.

```
use Algorithm::Permute;
my @array = 'a'..'d';
my $p_iterator = Algorithm::Permute->new ( \@array );
while (my @perm = $p_iterator->next) {
  print "next permutation: (@perm)\n";
}
```

Pour une exécution encore plus rapide, vous pouvez faire :

```
use Algorithm::Permute;
my @array = 'a'..'d';
Algorithm::Permute::permute {
  print "next permutation: (@array)\n";
} @array;
```

Voici un petit programme qui, à chaque ligne entrée, génère toutes les permutations de des mots de la ligne. L'algorithme utilisé dans la fonction permute() est présenté dans le quatrième volume (non encore publié) de *The Art of Computer Programming* par Knuth et fonctionnera pour n'importe quelle liste :

```
#!/usr/bin/perl -n
# générateur de permutations ordonnées de Fischer-Kause

sub permute (&@) {
  my $code = shift;
  my @idx = 0..$#_;
  while ( $code->(@_[@idx]) ) {
    my $p = $#idx;
    --$p while $idx[$p-1] > $idx[$p];
    my $q = $p or return;
    push @idx, reverse splice @idx, $p;
    ++$q while $idx[$p-1] > $idx[$q];
    @idx[$p-1,$q]=@idx[$q,$p-1];
  }
}

permute {print"@_\n"} split;
```

6.14 Comment trier un tableau par (n'importe quoi) ?

Fournir une fonction de comparaison à sort() (décrit dans sort in *perlfunc*) :

```
@list = sort { $a <=> $b } @list;
```

La fonction de tri par défaut est cmp, la comparaison des chaînes, laquelle trierait `(1, 2, 10)` en `(1, 10, 2)`. <=>, utilisé ci-dessus, est l'opérateur de comparaison numérique.

Si vous utilisez une fonction compliquée pour extraire la partie sur laquelle vous souhaitez faire le tri, il vaut mieux ne pas le faire à l'intérieur de la fonction sort. L'extraire d'abord, parce que le BLOC de tri peut être appelé plusieurs fois pour un même élément. Voici par exemple comment récupérer le premier mot après le premier nombre de chaque élément, et ensuite faire le tri sur ces mots sans tenir compte de la casse des caractères.

```
@idx = ();
for (@data) {
    ($item) = /\d+\s*(\S+)/;
    push @idx, uc($item);
}
@sorted = @data[ sort { $idx[$a] cmp $idx[$b] } 0 .. $#idx ];
```

qui pourrait aussi s'écrire ainsi, en utilisant une astuce qui est connue sous le nom de Transformation Schwartzienne :

```
@sorted = map  { $_->[0] }
          sort { $a->[1] cmp $b->[1] }
          map  { [ $_, uc( (/\d+\s*(\S+)/)[0]) ] } @data;
```

Si vous avez besoin de trier sur plusieurs champs, le paradigme suivant est utile :

```
@sorted = sort { field1($a) <=> field1($b) ||
                 field2($a) cmp field2($b) ||
                 field3($a) cmp field3($b)
               }     @data;
```

Ceci pouvant être aisément combiné avec le calcul préalable des clefs comme détaillé ci-dessus.

Voir l'article *sort* dans la collection "Far More Than You Ever Wanted To Know" sur http://www.cpan.org/misc/olddoc/FMTEYEWTK.tgz pour plus détails sur cette approche.

Voir aussi ci-dessous la question relative au tri des tables de hachage.

6.15 Comment manipuler des tableaux de bits ?

Utiliser pack() et unpack(), ou encore vec() et les opérations bit à bit.

Par exemple, ceci impose à $vec d'avoir le bit N positionné si $ints[N] était positionné :

```
$vec = '';
foreach(@ints) { vec($vec,$_,1) = 1 }
```

Voici comment, avec un vecteur dans $vec, amener ces bits dans votre tableau d'entiers @ints :

```
 sub bitvec_to_list {
    my $vec = shift;
    my @ints;
    # Choisir le meilleur algorithme suivant la densité de bits nuls
    if ($vec =~ tr/\0// / length $vec > 0.95) {
        use integer;
        my $i;
        # Méthode rapide avec surtout des bits nuls
        while($vec =~ /[^\0]/g ) {
            $i = -9 + 8 * pos $vec;
```

```
                push @ints, $i if vec($vec, ++$i, 1);
                push @ints, $i if vec($vec, ++$i, 1);
                push @ints, $i if vec($vec, ++$i, 1);
                push @ints, $i if vec($vec, ++$i, 1);
                push @ints, $i if vec($vec, ++$i, 1);
                push @ints, $i if vec($vec, ++$i, 1);
                push @ints, $i if vec($vec, ++$i, 1);
                push @ints, $i if vec($vec, ++$i, 1);
                push @ints, $i if vec($vec, ++$i, 1);
            }
        } else {
            # Algorithme général rapide
            use integer;
            my $bits = unpack "b*", $vec;
            push @ints, 0 if $bits =~ s/^(\d)// && $1;
            push @ints, pos $bits while($bits =~ /1/g);
        }
        return \@ints;
    }
```

Cette méthode devient d'autant plus rapide que le vecteur en bits est clairsemé. (Avec la permission de Tim Bunce et Winfried Koenig.)

Vous pouvez raccourcir un peu la boucle grâce à cette suggestions de Benjamin Goldberg :

```
while($vec =~ /[^\0]+/g ) {
  push @ints, grep vec($vec, $_, 1), $-[0] * 8 .. $+[0] * 8;
}
```

Ou utiliser le module Bit::Vector du CPAN :

```
$vector = Bit::Vector->new($num_of_bits);
$vector->Index_List_Store(@ints);
@ints = $vector->Index_List_Read();
```

Bit::Vector fournit des méthodes efficaces pour gérer des vecteurs de bits, de petit entiers ou de "big int".

Voici un exemple plus complet d'utilisation de vec() :

```
    # démo de vec
    $vector = "\xff\x0f\xef\xfe";
    print "Ilya's string \\xff\\x0f\\xef\\xfe represents the number ",
        unpack("N", $vector), "\n";
    $is_set = vec($vector, 23, 1);
    print "Its 23rd bit is ", $is_set ? "set" : "clear", ".\n";
    pvec($vector);

    set_vec(1,1,1);
    set_vec(3,1,1);
    set_vec(23,1,1);

    set_vec(3,1,3);
    set_vec(3,2,3);
    set_vec(3,4,3);
    set_vec(3,4,7);
    set_vec(3,8,3);
    set_vec(3,8,7);

    set_vec(0,32,17);
    set_vec(1,32,17);
```

```
sub set_vec {
    my ($offset, $width, $value) = @_;
    my $vector = '';
    vec($vector, $offset, $width) = $value;
    print "offset=$offset width=$width value=$value\n";
    pvec($vector);
}

sub pvec {
    my $vector = shift;
    my $bits = unpack("b*", $vector);
    my $i = 0;
    my $BASE = 8;

    print "vector length in bytes: ", length($vector), "\n";
    @bytes = unpack("A8" x length($vector), $bits);
    print "bits are: @bytes\n\n";
}
```

6.16 Pourquoi defined() retourne vrai sur des tableaux et hachages vides ?

La petite histoire est que vous ne devriez probablement utiliser defined que sur les scalaires ou les fonctions, et pas sur les agrégats (tableaux et hachages). Voir defined in *perlfunc* dans les versions 5.004 et suivantes de Perl pour plus de détails.

7 Données : tables de hachage (tableaux associatifs)

7.1 Comment traiter une table entière ?

Utiliser la fonction each() (voir each in *perlfunc*) si vous n'avez pas besoin de la trier :

```
while ( ($key, $value) = each %hash) {
    print "$key = $value\n";
}
```

Si vous la souhaitez triée, il vous faudra utiliser foreach() sur le résultat du tri des clefs, comme montré dans une question précédente.

7.2 Que se passe-t-il si j'ajoute ou j'enlève des clefs d'un hachage pendant que j'itère dessus ?

(contribution de brian d foy)

La réponse simple est « Ne le faites pas ! »

Si vous parcourez la table de hachage via each(), la seule clé que vous pouvez supprimer sans problème est la clé courante. Si vous supprimez ou ajoutez d'autres clés, l'itérateur pourrait oublier certaines clés ou présenter deux fois la même clé puisque perl risque de réarranger la table de hachage. Voir `each()` dans *perlfunc*.

7.3 Comment rechercher un élément d'un hachage par sa valeur ?

Créer un hachage inversé :

```
%by_value = reverse %by_key;
$key = $by_value{$value};
```

Ceci n'est pas particulièrement efficace. Il aurait été plus efficace pour l'espace de stockage d'utiliser :

```
while (($key, $value) = each %by_key) {
    $by_value{$value} = $key;
}
```

Si votre hachage pouvait avoir plus d'une valeur identique, les méthodes ci-dessus ne trouveront qu'une seule des clefs associées. Ceci peut être ou ne pas être un problème pour vous. Si cela vous inquiète, vous pouvez toujours inverser le hachage en un hachage de tableaux :

```
while (($key, $value) = each %by_key) {
    push @{$key_list_by_value{$value}}, $key;
}
```

7.4 Comment savoir combien d'entrées sont dans un hachage ?

Si vous voulez dire combien de clefs, alors il vous suffit de prendre dans son sens scalaire la fonction keys() :

```
$num_keys = keys %hash;
```

La fonction keys() réinitialise les itérateurs de la table de hachage, ce qui signifie que vous pourriez avoir des résultats étranges si vous l'utilisez pendant l'utilisation sur la même table d'autres opérateurs de table de hachage tels que each().

7.5 Comment trier une table de hachage (par valeur ou par clef) ?

(contribution de brian d foy)

Pour trier une table de hachage, commençons par nous intéresser aux clés. Dans l'exemple suivant, nous fournissons la liste des clés à la fonction sort qui les trie ASCIIbétiquement (cet ordre peut-être modifié par vos réglages de locale). Une fois les clés récupérées dans le bon ordre, nous pouvons les utiliser pour afficher un rapport listant ces clés dans l'ordre ASCIIbétique.

```
my @keys = sort { $a cmp $b } keys %hash;

foreach my $key ( @keys )
  {
    printf "%-20s %6d\n", $key, $hash{$value};
  }
```

Nous pouvons utiliser le bloc de `sort()` d'une manière plus puissante. Au lieu de comparer directement les clés, nous pouvons calculer une valeur à partir de chaque clé et utiliser ces valeurs dans la comparaison.

Par exemple, pour rendre notre ordre d'affichage insensible à la casse, nous utilisons la séquence `\L` dans une chaîne entre guillemets pour tout transformer en minuscules. Le bloc `sort()` comparera donc les valeurs des clés en minuscules pour les ordonner.

```
my @keys = sort { "\L$a" cmp "\L$b" } keys %hash;
```

Note : si la transformation des valeurs des clés est coûteuse ou si la table a beaucoup d'éléments, vous pouvez jeter un oeil à la Transformation Schwartzienne pour placer le résultat de la transformation en cache.

Si nous voulons trier notre table de hachage selon l'ordre de ses valeurs, nous utilisons les clés pour y accéder. Nous obtenons toujours une liste de clés mais ordonnée selon les valeurs.

```
my @keys = sort { $hash{$a} <=> $hash{$b} } keys %hash;
```

À partir de là, nous pouvons faire des choses plus complexes. Par exemple, si les valeurs sont identiques, nous pouvons alors utiliser les clés comme ordre secondaire.

```
my @keys = sort {
                 $hash{$a} <=> $hash{$b}
                           or
                 "\L$a" cmp "\L$b"
                } keys %hash;
```

7.6 Comment conserver mes tables de hachage dans l'ordre ?

Vous pouvez regarder dans le module DB_File et attacher avec tie() en utilisant les liens de hachage $DB_TREE tel que documenté dans DB_File/"In Memory Databases". Le module Tie::IxHash du CPAN peut aussi être instructif.

7.7 Quelle est la différence entre "delete" et "undef" pour des tables de hachage ?

Les table de hachage contiennent des paires de `scalaires` : le premier est la clef, le second est la valeur. La clef sera toujours convertie en une chaîne alors que la valeur peut être n'importe quelle sorte de scalaire : chaîne, nombre ou référence. Si la clef `$key` est présente dans la table, `exists($hash{$key})` renverra vrai. La valeur associée à une clef donnée peut être `undef`, auquel cas `$hash{$key}` vaudra `undef` tandis que `exists $hash{$key}` retournera vrai. Ceci correspond au fait d'avoir dans la table de hachage la paire (`$key`, `undef`).

Les dessins aident... Voici la table de hachage %hash :

```
  clefs  valeurs
+-------+-------+
|   a   |   3   |
|   x   |   7   |
|   d   |   0   |
|   e   |   2   |
+-------+-------+
```

La valeur de vérité de quelques conditions est alors :

```
$hase{'a'}                    est vrai
$hash{'d'}                    est faux
defined $hash{'d'}            est vrai
defined $hash{'a'}            est vrai
exists $hash{'a'}             est vrai (Perl5 seulement)
grep ($_ eq 'a', keys %hash)  est vrai
```

Si vous dites maintenant :

```
undef $ary{'a'}
```

La table devient la suivante :

```
  clefs  valeurs
+-------+-------+
|   a   | undef |
|   x   |   7   |
|   d   |   0   |
|   e   |   2   |
+-------+-------+
```

Et voici les nouvelles valeurs de vérité de nos conditions (les changements sont en majuscules) :

```
$hash{'a'}                    est FAUX
$hash{'d'}                    est faux
defined $hash{'d'}            est vrai
defined $hash{'a'}            est FAUX
exists $hash{'a'}             est vrai (Perl5 seulement)
grep ($_ eq 'a', keys %hash)  est vrai
```

Remarquez les deux dernières : on a une valeur undef, mais une clef définie !

Maintenant considérez ceci :

```
delete $hash{'a'}
```

La table se lit maintenant :

```
  clefs  valeurs
+-------+-------+
|   x   |   7   |
|   d   |   0   |
|   e   |   2   |
+-------+-------+
```

Et nos nouvelles valeurs de vérité sont (avec les changements en majuscules) :

```
$hash{'a'}                    est faux
$hash{'d'}                    est faux
defined $hash{'d'}            est vrai
defined $hash{'a'}            est faux
exists $hash{'a'}             est FAUX (Perl5 seulement)
grep ($_ eq 'a', keys %hash)  est FAUX
```

Voyez, tout ce qui est lié à cette clé a disparu !

7.8 Pourquoi mes tables de hachage liées (par tie()) ne font pas la distinction entre exists et defined ?

Cela dépend de l'implémentation de EXISTS() à laquelle est attachée la table de hachage. Par exemple, les tables liées à des fichiers DBM* ne connaissent pas la notion de valeur undef. Cela signifie donc que exists() et defined() font la même chose sur un fichier DBM* et que ce qu'elles font ne correspond pas à ce qu'elles feraient sur une table de hachage ordinaire.

7.9 Comment réinitialiser une opération each() non terminée ?

L'utilisation de `keys %hash` dans un contexte scalaire renvoie le nombre de clefs d'un hachage et réinitialiser les itérateurs associés à la table de hachage. Vous pouvez en avoir besoin lorsque vous sortez prématurément d'une boucle avec un `last` afin de réinitialiser l'itérateur de cette boucle lorsque vous voudrez la réutiliser.

7.10 Comment obtenir l'unicité des clefs de deux hachages ?

Tout d'abord, extraire les clefs des hachages dans des listes, puis résoudre le problème de la "suppression des doublons" tel que décrit plus haut. Par exemple :

```
%dejavu = ();
for $element (keys(%foo), keys(%bar)) {
    $dejavu{$element}++;
}
@uniq = keys %dejavu;
```

Ou plus succinctement :

```
@uniq = keys %{{%foo,%bar}};
```

Ou, pour vraiment économiser de la place :

```
%dejavu = ();
while (defined ($key = each %foo)) {
    $dejavu{$key}++;
}
while (defined ($key = each %bar)) {
    $dejavu{$key}++;
}
@uniq = keys %dejavu;
```

7.11 Comment enregistrer un tableau multidimensionnel dans un fichier DBM ?

Soit vous transformez vous-mêmes la structure en une chaîne de caractères (casse-tête), soit vous récupérez le module MLDBM (qui utilise Data::Dumper) du CPAN, et vous l'utilisez au-dessus de DB_File ou GDBM_File.

7.12 Comment faire en sorte que mon hachage conserve l'ordre des éléments que j'y mets ?

Utiliser le module Tie::IxHash du CPAN.

```
use Tie::IxHash;
tie my %myhash, 'Tie::IxHash';
for (my $i=0; $i<20; $i++) {
  $myhash{$i} = 2*$i;
}
my @keys = keys %myhash;
# @keys = (0,1,2,3,...)
```

7.13 Pourquoi le passage à un sous-programme d'un élément non défini d'un hachage le crée du même coup ?

Si on dit quelque chose comme :

```
somefunc($hash{"nonesuch key here"});
```

Alors cet élément "s'autoactive" ; c'est-à-dire qu'il se crée tout seul, que l'on y range quelque chose ou pas. Ceci arrive parce que les fonctions reçoivent des scalaires passés par références. Si somefunc() modifie `$_[0]`, elle doit être prête à la réécrire dans la version de l'appelant.

Ceci a été réglé à partir de Perl5.004.

Normalement, d'accéder simplement à la valeur d'une clef dans le cas d'une clef inexistante ne produit *pas* une clef présente éternellement. Ceci est différent du comportement d'awk.

7.14 Comment faire l'équivalent en Perl d'une structure en C, d'une classe/d'un hachage en C++ ou d'un tableau de hachages ou de tableaux ?

Habituellement via une référence de hachage, peut-être comme ceci :

```
$record = {
    NAME   => "Jason",
    EMPNO  => 132,
    TITLE  => "deputy peon",
    AGE    => 23,
    SALARY => 37_000,
    PALS   => [ "Norbert", "Rhys", "Phineas"],
};
```

Les références sont documentées dans *perlref* et le futur *perlreftut*. Des exemples de structures de données complexes sont données dans *perldsc* et *perllol*. Des exemples de structures et de classes orientées objet sont dans *perltoot*.

7.15 Comment utiliser une référence comme clef d'une table de hachage ?

(contribution de brian d foy)

Les clés des tables de hachage sont des chaînes donc vous ne pouvez pas vraiment utiliser une référence comme clé. Si vous essayez de le faire, perl transforme la référence en une chaîne (par exemple `HASH(0xDEADBEEF)`). Vous ne pouvez pas retrouver la référence originale à partir de cette chaîne, tout du moins sans fournir vous-même un travail supplémentaire. Souvenez-vous aussi que les clés sont uniques mais deux variables différentes peuvent contenir la même référence (et ces variables peuvent ensuite changer).

La module Tie::RefHash, qui est distribué avec perl, est peut-être ce que vous cherchez. Il gère le travail supplémentaire.

8 Données : divers

8.1 Comment manipuler proprement des données binaires ?

Perl est adapté au binaire, donc ceci ne devrait pas être un problème. Par exemple, ceci marche correctement (en supposant les fichiers trouvés) :

```
if (`cat /vmunix` =~ /gzip/) {
    print "Your kernel is GNU-zip enabled!\n";
}
```

Sur les systèmes moins élégants (lire : byzantins), on doit cependant jouer à des jeux éreintants en séparant les fichiers textes ("text") des fichiers binaires ("binary"). Voir binmode in *perlfunc* ou *perlopentut*.

Si le problème provient de données ASCII 8-bits, alors voir *perllocale*.

Si vous souhaitez agir sur des caractères multi-octets, alors il y a quelques pièges. Voir la section sur les expressions régulières.

8.2 Comment déterminer si un scalaire est un nombre/entier/à virgule flottante ?

En supposant que vous ne vous occupiez pas des notations IEEE comme "NaN" ou "Infinity", il vous suffit probablement d'utiliser une expression régulière.

```
if (/\D/)            { print "has nondigits\n" }
if (/^\d+$/)         { print "is a whole number\n" }
if (/^-?\d+$/)       { print "is an integer\n" }
if (/^[+-]?\d+$/)    { print "is a +/- integer\n" }
if (/^-?\d+\.?\d*$/) { print "is a real number\n" }
if (/^-?(?:\d+(?:\.\d*)?|\.\d+)$/) { print "is a decimal number\n" }
if (/^([+-]?)(?=\d|\.\d)\d*(\.\d*)?([Ee]([+-]?\d+))?$/)
                     { print "a C float\n" }
```

Il existe aussi quelques modules communs pour faire cela. *Scalar::Util* (distribué avec perl 5.8) donne accès à la fonction interne de perl `looks_like_number` qui permet de déterminer si une valeur ressemble à un nombre. *Data::Types* exporte des fonctions qui valident le type de donnée en utilisant les expressions ci-dessus ainsi que d'autres expressions rationnelles. Il y a aussi *Regexp::Common* qui propose des expressions rationnelles pour reconnaître différents types de nombres. Ces trois modules sont disponibles sur CPAN.

Sur un système POSIX, Perl accepte la fonction `POSIX::strtod`. Sa sémantique est quelque peu malcommode, donc il y a une fonction d'emballage `getnum` pour un usage plus aisé. Cette fonction prend une chaîne et retourne le nombre qu'elle a trouvé, ou `undef` pour une entrée qui n'est pas un nombre à virgule flottante du C. La fonction `is_numeric` est une couverture frontale à `getnum`, au cas ou vous voudriez simplement demander « Ceci est-il un nombre à virgule flottante ? »

```
 sub getnum {
    use POSIX qw(strtod);
    my $str = shift;
    $str =~ s/^\s+//;
    $str =~ s/\s+$//;
    $! = 0;
    my($num, $unparsed) = strtod($str);
    if (($str eq '') || ($unparsed != 0) || $!) {
        return undef;
    } else {
        return $num;
    }
}

sub is_numeric { defined &getnum($_[0]) }
```

À la place, vous pouvez également regarder le module *String::Scanf* sur CPAN. Le module POSIX (qui fait partie de la distribution standard de Perl) propose les fonctions `strtol` et `strtod` pour convertir des chaînes respectivement en longs et en doubles.

8.3 Comment conserver des données persistantes entre divers appels de programme ?

Pour des application spécifiques, on peut utiliser l'un des modules DBM. Voir AnyDBM_File. Plus généralement, vous devriez consulter les modules FreezeThaw ou Storable du CPAN. Depuis Perl 5.8, Storable fait partie de la distribution standard. Voici un exemple utilisant les fonctions `store` et `retrieve` de Storable :

```
use Storable;
store(\%hash, "filename");

# later on...
$href = retrieve("filename");        # par référence
%hash = %{ retrieve("filename") };   # accès direct au hachage
```

8.4 Comment afficher ou copier une structure de données récursive ?

Le module Data::Dumper du CPAN (ou en standard depuis Perl 5.005) est agréable pour afficher les structures de données. Le module Storable sur CPAN (ou en standard depuis Perl 5.8) fournit une fonction appelée `dclone` qui copie récursivement ses arguments.

```
use Storable qw(dclone);
$r2 = dclone($r1);
```

Où $r1 peut être une référence à tout type de structure de données. Il sera copié en profondeur. Puisque `dclone` prend et renvoie des références, vous devez ajouter de la ponctuation si c'est un hachage de tableaux que vous désirez copier.

```
%newhash = %{ dclone(\%oldhash) };
```

8.5 Comment définir des méthodes pour toutes les classes ou tous les objets ?

Utiliser la classe UNIVERSAL (voir *UNIVERSAL*).

8.6 Comment vérifier la somme de contrôle d'une carte de crédit ?

Récupérer le module Business::CreditCard du CPAN.

8.7 Comment compacter des tableaux de nombres à virgule flottante simples ou doubles pour le code XS ?

Le code kgbpack.c dans le module PGPLOT du CPAN fait pile cela. Si vous faites beaucoup de traitement de nombres à virgule flottante, vous pouvez envisager d'utiliser à la place le module PDL du CPAN – il rend la chose plus facile.

9 AUTEUR ET COPYRIGHT

10 TRADUCTION

La traduction française est distribuée avec les même droits que sa version originale (voir ci-dessus).

10.1 Version

Cette traduction française correspond à la version anglaise distribuée avec perl 5.8.8. Pour en savoir plus concernant ces traductions, consultez http://perl.enstimac.fr/.

10.2 Traducteur

Traduction initiale : Guillaume van der Rest <gvdr@dcmr.polytechnique.fr>. Mise à jour : Roland Trique <roland.trique@uhb.fr>, Paul Gaborit <paul.gaborit at enstimac.fr>

10.3 Relecture

Régis Julié <Regis.Julie@cetelem.fr>, Gérard Delafond

11 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.