perlnumber

Table des matières

1 NAME/NOM

perlnumber - Sémantique des nombres et opérations numériques en Perl

2 SYNOPSIS

```
$n = 1234;              # entier en décimal
$n = 0b1110011;         # entier en binaire
$n = 01234;             # entier en octal
$n = 0x1234;            # entier en hexadecimal
$n = 12.34e-56;         # notation exponentielle
$n = "-12.34e56";       # nombre spécifié sous forme de chaîne
$n = "1234";            # nombre spécifié sous forme de chaîne
```

3 DESCRIPTION

Ce document décrit comment Perl manipule les valeurs numériques en interne.

Les possibilités de surcharge d'opérateurs en Perl sont complètement ignorées ici. La surcharge d'opérateurs permet à l'utilisateur d'adapter le comportement des nombres pour obtenir, par exemple, des opérations sur des entiers ou des flottants en précision arbitraire ou des opérations sur des nombres "exotiques" telles l'arithmétique modulaire ou l'arithmétique p-adique et ainsi de suite. Voir *overload* pous plus de détails.

4 Stockage des nombres

Perl, en interne, peut représenter un nombre de trois manières différentes : comme un entier natif, comme un flottant natif et comme une chaîne de caractères en décimal. Les chaînes de caractères en décimal peuvent contenir un exposant, comme dans `"12.34e-56"`. *Natif* signifie ici "un format reconnu par le compilateur C qui a été utilisé pour construire perl".

Le terme "natif" n'a pas autant d'implications lorsqu'on parle d'entiers natifs que quand on parle de flottants natifs. La seule implication du terme "natif" appliqué aux entiers est que les limites maximales et minimales acceptées pour une quantité entière sont très proches d'une puissance de 2. En revanche, les flottants "natifs" ont plus de restrictions : ils ne peuvent représenter que des nombres qui ont une représentation relativement "courte" lorsqu'ils sont convertis en base 2. Par exemple, 0.9 ne peut pas être représenté par un flottant natif puisque son écriture en base 2 est infinie :

```
0.1110011001100...
```

avec la séquence `1100` qui se répète indéfiniment ensuite. En plus de la limitation précédente, la partie exposant du nombre est, elle aussi, limitée lorsqu'elle est représentée par un flottant. Sur la plupart des matériels, les valeurs flottantes peuvent stocker des nombres contenant au plus 53 chiffres binaires avec une partie exposant comprise entre -1024 et 1024. Une fois ramenées en décimale, ces limites donnent environ 16 chiffres significatifs et une puissance de dix comprises entre -304 et 304. Une conséquence de tout cela est donc que, sur de telles architectures, Perl ne peut pas stocker dans un flottant un nombre comme 12345678901234567 sans perdre d'information.

De manière similaire, les chaînes de caractères en décimal ne peuvent représenter que des nombres dont la représentation décimale est finie. Comme ce sont des chaînes, et donc de longueur quelconque, il n'y a pratiquement pas de limite pour l'exposant ou le nombre de décimales de tels nombres. (Mais n'oubliez pas que nous ne parlons que des règles de *stockage* de ces nombres. Que vous puissiez effectivement stocker des "grands" nombres n'implique pas que vous pourrez réaliser des *opérations* sur ces nombres utilisant tous les chiffres disponibles. Voir Opérateurs et conversions numériques (§5) pour plus de détails.)

Dans les faits, les nombres stockés au format natif entier peuvent l'être sous leur forme avec ou sans signe. Donc, en Perl, les limites pour les entiers stockés au format natif sont typiquement -2**31..2**32-1, valeurs à adapter dans le cas des entiers sur 64 bits. Encore une fois, cela ne signifie pas que Perl peut faire des opérations uniquement sur des entiers dans cet intervalle : il est possible de stocker bien plus d'entiers en utilisant le format flottant.

En résumé, les valeurs numériques que Perl peut stocker sont celles qui ont une représentation décimale finie ou une "courte" représentation binaire.

5 Opérateurs et conversions numériques

Perl peut stocker un nombre dans n'importe lequel des trois formats mentionnés ci-dessus mais, typiquement, la plupart des opérateurs ne comprennent qu'un seul de ces formats. Lorsqu'une valeur numérique est passée comme argument à l'un de ces opérateurs, elle est convertie vers le format compris par l'opérateur.

Six conversions sont donc possibles :

```
entier natif     -->  flottant natif   (*)
entier natif     -->  chaîne décimale
flottant natif   -->  entier natif     (*)
flottant natif   -->  chaîne décimale  (*)
chaîne décimale  -->  entier natif
chaîne décimale  -->  flottant natif   (*)
```

Ces conversions suivent les règles générales suivantes :

- Si le nombre d'origine peut être représenté sous la forme visée, cette représentation est utilisée.
- Si le nombre d'origine est en dehors des limites représentables par la forme visée, la représentation choisie est celle de la limite la plus proche. (*perte d'information*)
- Si le nombre d'origine est compris entre deux nombres représentables par la forme visée, la représentation de l'un de ces deux nombres est utilisée. (*perte d'information*)
- Lors de la conversion `flottant natif -> entier natif`, la partie entière du résultat est toujours inférieure ou égale à la partie entière de la valeur d'origine. (*"arrondi vers zéro"*)
- Si la conversion `chaîne décimale -> entier natif` ne peut pas être réalisée sans perte d'information, le résultat est compatible avec la suite de conversions `chaîne décimale -> flottant natif -> entier natif`. En particulier, cela donne une chance à un nombre tel que `"0.99999999999999999"` d'être arrondi à 1.

RESTRICTION : les conversions ci-dessus marquées d'une `*` sont en partie effectués par le compilateur C. Donc, des bogues ou des particularités du compilateur utilisé peuvent parfois amener au non respect de certaines des règles ci-dessus.

6 Un avant-goût des opérations numériques en Perl

Les opérations Perl qui prennent un argument numérique traitent cet argument selon l'une des quatre manières suivantes : soit elles le transforment vers l'un des trois formats entier, flottant ou chaîne décimale, soit elles se comportent différement selon le format d'origine de l'argument. La conversion de la valeur numérique vers un format particulier ne change pas le nombre stocké dans la valeur.

Tous les opérateurs qui nécessitent un argument au format entier traitent leur argument en arithmétique modulaire, par exemple `mod 2**32` sur une architecture 32-bits. `spintf "%u", -1` fournira donc le même résultat que `spintf "%u", ~0`.

Les opérateurs arithmétiques

Les opérateurs binaires `+`, `-`, `*`, `/`, `%`, `==`, `!=`, `>`, `<`, `>=` et `<=` ainsi que les opérateurs unaires `-`, `abs` et `-` essayeront de convertir leurs arguments en entiers. Si les deux conversions sont possibles sans perte de précision et si l'opération peut être effectuée sans perte de précision alors un résultat entier sera produit. Sinon les arguments sont convertis en flottant et le résultat sera un flottant. Les conversions utilisant une sorte de cache (comme décrit ci-dessous), les conversions vers des entiers ne perdront pas la partie décimale des nombres flottants.

++

`++` se comportent comme les opérateurs ci-dessus sauf si son argument est une chaîne de caractères qui est reconnue par l'expression rationnelle `/^[a-zA-Z]*[0-9]*\z/`. Dans ce dernier cas, c'est l'incrémentation de chaîne décrite dans *perlop* qui est utilisée.

Les opérateurs arithmétiques lorsque `use integrer` est actif

Si `use integrer;` est actif, la quasi totalité des opérateurs listés ci-dessus convertissent leur(s) argument(s) au format entier et retourne un résultat entier. Les exceptions sont `abs`, `++` et `-` qui ne changent pas leur comportement.

Les autres opérateurs mathématiques

Les opérateurs tels que `**`, `sin` et `exp` convertissent leurs arguments vers le format flottant.

Les opérateurs bit-à-bit

Les arguments sont convertis au format entier si ce ne sont pas des chaînes de caractères.

Les opérateurs bit-à-bit lorsque `use integrer` est actif

Les arguments sont convertis au format entier. De plus les opérations internes de décalage utilisent des entiers signés au lieu des non signés par défaut.

Les opérations qui attendent un entier

L'argument est converti au format entier. Ceci s'applique, par exemple, au troisième et au quatrième argument de `sysread`.

Les opérations qui attendent une chaîne

L'argument est converti au format chaîne décimale. Par exemple, cela s'applique à <printf "%s", $value>.

Bien que la conversion d'un argument vers un format particulier ne change pas le nombre stocké, Perl se souvient du résultat de ces conversions. En particulier, bien que la première conversion puisse prendre du temps, des opérations répétées n'auront plus à refaire cette conversion.

7 AUTEUR

Ilya Zakharevich `ilya@math.ohio-state.edu`

Quelques adaptations par Gurusamy Sarathy <gsar@ActiveState.com>

Mise à jour pour 5.8.0 par Nicholas Clark <nick@ccl4.org>

8 VOIR AUSSI

overload et *perlop*.

9 TRADUCTION

9.1 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/.

9.2 Traducteur

Paul Gaborit (Paul.Gaborit at enstimac.fr).

9.3 Relecture

Personne pour l'instant.

10 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.