

Perl-Praxis

Komplexe Datenstrukturen

Jörn Clausen

`joern@TechFak.Uni-Bielefeld.DE`

Übersicht

- Hashes
- Referenzen
- komplexe Datenstrukturen

Hashes

- dritter grundlegender Datentyp von Perl
- Paare aus *keys* und *values*
- Notation: `%hash` ganzes Hash, `$hash{key}` einzelner Wert
- Schlüssel ist eindeutiger String
- Zuweisung einzelner Paare:

```
$days{jan} = 31;
```

```
$days{feb} = 28;
```

```
$days{mar} = 31;
```

- Verwendung:

```
$hours = ($days{jan} + $days{feb}) * 24;
```

Aufgaben

- Wieviele Schlüssel enthält das folgende Hash:

```
$hash{1}      = 1;
```

```
$hash{1.0}    = 2;
```

```
$hash{'1'}    = 3;
```

```
$hash{'1.0'}  = 4;
```

Definition von Hashes

- Zuweisung durch Liste:

```
%days = ( 'jan', 31, 'feb', 28, 'mar', 31 );
```

- schlecht lesbar
- schlecht erweiterbar, fehleranfällig
- besser:

```
%days = ( jan => 31,  
          feb => 28,  
          mar => 31 );
```

- => impliziert quotes um Schlüssel

Verwendung von Hashes

- Liste aller Schlüssel eines Hashes:

```
@months = keys(%days);
```

- typische Verwendung:

```
foreach $mon (keys(%days)) {  
    print "$mon has $days{$mon} days\n";  
}
```

- Reihenfolge der Schlüssel undefiniert
- lexikographisch sortiert:

```
foreach $mon (sort(keys(%days))) { ... }
```

- `values` liefert Liste aller Werte

Aufgaben

- Auch aus Hashes kann man *slices* extrahieren. Welcher *funny character* gehört an die Stelle des Fragezeichens?

```
?days{jan,mar}
```

- Lies die Datei `/etc/passwd` ein und extrahiere jeweils den Account-Namen und den tatsächlichen Namen der dort eingetragenen Benutzer. Speichere die Informationen in einem Hash und gib anschließend eine sortierte Liste aller Benutzer aus.

Verwendung von Hashes, cont.

- alternative Methode:

```
while (($month, $days) = each(%days)) {  
    print "$month has $days days\n";  
}
```

- Zuweisung an Skalar liefert nur Schlüssel:

```
$month = each(%days);
```

- `each`, `keys`, `values` benutzen selben Zähler
- Verhalten bei Veränderung des Hashes undefiniert

Hash als Menge

- Benutze nur Schlüssel, ignoriere Werte
- 6 aus 49, ohne Zusatzzahl:

```
while (keys(%numbers) < 6) {  
    $randnum = int(rand(49)) + 1;  
    $numbers{$randnum} = 1;  
}  
print join(", ", keys(%numbers)), "\n";
```

Schlüssel oder Wert?

- Arten des "Vorhandenseins":

```
%hash = ( a => 1,  
          b => 0,  
          c => undef );  
foreach $x ('a','b','c','d') {  
    print "$x is true\n"      if $hash{$x};  
    print "$x is defined\n"  if defined($hash{$x});  
    print "$x exists\n"     if exists($hash{$x});  
}
```

- Wert und Schlüssel löschen:

```
delete($hash{a});
```

Aufgaben

- Zähle die Häufigkeit der Wörter in einem beliebigen Text (z.B. den beiden Theaterstücken). Gib eine sortierte Liste der Wörter aus, mit den häufigsten Wörtern oben und den seltensten Wörtern unten.
- Ändere das Programm so ab, daß die Häufigkeit der Buchstaben gezählt wird.
- Ist es möglich, einem Schlüssel eine Liste von Werten zuzuordnen?
Was passiert hier:

```
%hash = ( a => (1, 2, 3),  
          b => (4, 5, 6),  
          c => (7, 8, 9) );
```

Referenzen

- Verweise auf Daten
- ähnlich zu Zeigern, aber einfacher
- keine Pointer-Arithmetik
- zwei Arten:
 - symbolische Referenzen
 - „echte“ Referenzen
- analog zu *symbolic links* und *hard links*
- Grundlage komplexer Datenstrukturen

Referenzen, cont.

- Referenzen definieren:

```
$scalar = 'Joe User';  
@array  = (1, 2, 3, 4, 5);  
%hash   = (a=>1, b=>2);
```

```
$scalar_ref = \ $scalar;  
$array_ref  = \ @array;  
$hash_ref   = \ %hash;
```

- Referenz selbst ist Skalar

- Dereferenzierung:

```
$name = $$scalar_ref;  
@nums = @$array_ref;  
%rel  = %$hash_ref;
```

```
$num = $$array_ref[2];  
$val = $$hash_ref{a};
```

Aufgaben

- Überzeuge Dich davon, daß man eine Referenz auf ein Array bilden kann:

```
@array = (1, 2, 3, 4, 5);  
$array_ref = \@array;
```

und diese anschließend wieder dereferenzieren kann:

```
foreach $num (@$array_ref) ...
```

- Welchen Wert hat `$array_ref`?
- Was passiert, wenn Du den „falschen“ funny character zur Dereferenzierung verwendest:

```
$what = $$array_ref;  
@what = @$hash_ref;
```

Referenzen, cont.

- alternative Notation:

```
$num = $array_ref->[2];  
$val = $hash_ref->{a};
```

- funktioniert auch über mehrere Ebenen:

```
$scalar_ref_ref = \ $scalar_ref;  
$name = $$$scalar_ref_ref;
```

- anonyme Referenzen:

```
$array_ref = [1, 2, 3, 4, 5];  
$hash_ref = {a=>1, b=>2};
```

komplexe Datenstrukturen

- Hash von Listen:

```
%hash = ( a => [1, 2, 3],  
          b => [4, 5, 6],  
          c => [7, 8, 9] );
```

- Zugriff auf einen Hash-Wert (eine Liste):

```
$array_ref = $hash{b};  
$num = $array_ref->[1];
```

- partielle Dereferenzierung:

```
@a = @$hash{a};      # funktioniert nicht  
@a = @{$hash{a}};
```

komplexe Datenstrukturen, cont.

- Zugriff auf ein Element in einer Liste:

```
$val = $hash{b}->[2];
```

- -> kann zwischen Klammern weggelassen werden:

```
$val = $hash{b}[2];
```

- beachte: %hash echtes Hash, daher kein -> vor erster Klammer
- und: -> nicht mit => verwechseln

LoLs, ...

- *list of lists:*

```
@atoms = ( ['Wasserstoff', 'H', 1],  
           ['Helium',      'He', 4],  
           ['Lithium',     'Li', 6] );
```

- Zugriff:

```
$atoms[1]->[2]  
$atoms[0][1]
```

... HoHs, ...

- *hash of hashes:*

```
%atoms = ( H => { name => 'Wasserstoff',  
               mass => 1 },  
          He => { name => 'Helium',  
               mass => 4 },  
          Li => { name => 'Lithium',  
               mass => 6 } );
```

- Zugriff:

```
$atoms{He}->{mass}  
$atoms{Li}{name}
```

... und Kombinationen

- *hash of lists:*

```
%atoms = ( H => ['Wasserstoff', 1],  
           He => ['Helium', 4],  
           Li => ['Lithium', 6] );
```

- *list of hashes:*

```
@atoms = ( { name => 'Wasserstoff',  
            symbol => 'H', mass => 1 },  
           { name => 'Helium',  
            symbol => 'He', mass => 4 },  
           { name => 'Lithium',  
            symbol => 'Li', mass => 6 } );
```

dynamische Datenstrukturen

- komplexe Datenstrukturen dynamisch erzeugen:

```
@names = ('Wasserstoff', 'Helium', 'Lithium');  
@symbols = ('H', 'He', 'Li');  
@masses = (1, 4, 6);
```

```
while (@names) {  
    $name = shift(@names);  
    $symbol = shift(@symbols);  
    $mass = shift(@masses);  
    push(@atoms, { name => $name,  
                  symbol => $symbol,  
                  mass => $mass } );  
}
```

Visualisierung

- zur Fehlersuche hilfreich:

```
use Data::Dumper;  
%atoms = ...  
print Dumper \%atoms;
```

- use sehen wir uns später genauer an

- Ausgabe:

```
$VAR1 = {  
    'H' => [  
        'Wasserstoff',  
        1  
    ],  
    ...  
}
```

Aufgaben

- Der Befehl „`ypcat group`“ gibt den Inhalt der im NIS gespeicherten Gruppen-Informationen aus. Parse die Ausgabe und speichere die Informationen in den Hashes `%groups` und `%users`. Es soll folgendes möglich sein:

```
$gid      = $groups{www}{gid};  
@members = @{$groups{www}{members}};  
@groups  = @{$users{joern}};
```