

# Komplexe Datenstrukturen in Perl

**Seminar: Angewandte computerlinguistische Programmier-Techniken mit Perl**

Dozent: David Reitter  
Referent: Sven Brüßow

## Arrays - (1)

```
#!/usr/bin/perl -w
@List = ("TaCoS ",2002,"Potsdam");
print "\n". "Die ".$List[0], " findet ", $List[1];
print " in ", $List[2], " statt.".\n";
```

- Arrays (Listen) sind Ketten von **Skalaren**.
- Sie werden mit dem Klammeraffen **@** eingeleitet
- Jedes Element hat einen **Index**, über den auf das entsprechende Element zugegriffen werden kann.
- Die **Indexierung** beginnt mit **[0]**.
- `$List[0] = "TaCos "` (ist ein Skalar!)

## Arrays - (2)

```
@a=(4,5,6,'a','b','c');
print @a;           # 456abc
print @a[1,2];     # 56
print $a[1];       # 5

@zahlen = (1..10);
foreach(@zahlen) {
    print $_, "\n";
}
```

## Arrays - (3)

```
@satz=("jeder ","mann ","liebt ","eine ","frau.");
# gibt den ganzen array aus
print @satz, "\n";

# gibt anzahl der elemente aus
print scalar(@satz);

# gibt ebenfalls anzahl der elemente aus, da
# konkatenation skalaren kontext herstellt.
print @satz, "\n";

Ausgabe:
jeder mann liebt eine frau.
5
5
```

## Arrays - Zugriff (1)

```
#!/usr/bin/perl -w
@Letter = ("a".."z");
print $Letter[8], "\n";           # gibt 'i' aus

for $i ( 0 .. @Letter ) {        # gibt alles aus
    print $Letter[$i]."\n";
}
```

## Referenzen (1)

In Perl sind alle Datenstrukturen, ob sie nun ein- oder mehrdimensional sind, **intern immer eindimensional**. Das gilt also sowohl für Skalare, als auch für Arrays und Hashes.

Der Wert einer Datenstruktur ist immer ein Skalarwert (scalar values), also entweder ein String, eine Zahl oder eine **Referenz (Pointer)**.

```
@event = ([ "SuB99",1999,"Duesseldorf"],
          ["TaCoS ",2002,"Potsdam"]);

print @event;
```

**Ausgabe:**

```
ARRAY(0x1a7f184) ARRAY(0x1a755ec)
```

## Referenzen (2)

- Pointer werden in Perl **Referenzen** genannt.
- Unterstützt werden Referenzen seit Version 5.
- Mit Referenzen kann der Inhalt eines Skalars, eines Arrays oder eines Hashes wie über den eigentlichen Namen der angesprochen werden.
- **Referenzen können auch auf auf Subroutinen zeigen.**
- **Referenzen sind Skalare**, die aber keine Werte speichern, sondern **Arbeitsspeicheradressen** von anderen Variablen oder von Subroutinen

Es wird zwischen zwei Sorten unterschieden

- **Symbolische Referenzen**
- **Harte Referenzen**

## Referenzen - symbolische

Für symbolische Referenzen wird sei hier nur ein Beispiel angegeben. Im weiteren Verlauf werden ausschließlich harte Referenzen behandelt.

```
$var1 = "SuB99 "; # Ein ganz normaler Skalar
$var = 'var1'; # Eine symbolische Referenz
print "$var\n"; # var1
print "$$var\n"; # SuB99
$$var = "TaCoS"; # $var == "TaCoS"
print "$var1\n"; # TaCoS
```

**Ausgabe:**

```
var1
SuB99
TaCoS
```

## Referenzen auf Skalare

```
$satz = "Jeder Mann liebt eine Frau";
$satzref = \$satz;
print $$satzref, "\n"; #oder
print ${$satzref}, "\n";
print \$satz;
```

**Ausgabe:**

```
Jeder Mann liebt eine Frau
Jeder Mann liebt eine Frau
SCALAR(0x1a727bc)
```

Ohne den vorangestellten Backslash würde einfach nur eine Kopie von \$satzref angelegt werden, keine Referenz.

## Referenzen auf Arrays

```
@tags = ( "NN ", "NE ", "ADJA ", "ADV ", "VVFIN " );
$tagsref = \@tags; # Die Referenz ist ein Skalar!
$tagsref2 = @tags;
print $tagsref, "\n";
print $tagsref2, "\n";
print @$tagsref, "\n";
print \@tags, "\n";
print @tags, "\n";
print $tagsref->[2]; # ${$tagsref}[2] oder $$tagsref[2]
```

**Ausgabe:**

```
ARRAY(0x1a72978)
5
NN NE ADJA ADV VVFIN
ARRAY(0x1a72978)
NN NE ADJA ADV VVFIN
ADJA
```

## Referenzen auf Hashes (1)

```
%getaggt = ( NN => "Mann",
            NE => "Maria",
            ADJA => "rote",
            ADV => "schon",
            VVFIN => "gehst",
            VAFIN => "bist");
```

```
$ref = \%getaggt;
$ref2 = %getaggt;
print $ref, "\n";
print $ref2, "\n";
print %$ref, "\n";
print \@tags, "\n";
print $$ref{ADV};
```

## Referenzen auf Hashes (2)

**Ausgabe:**

```
HASH(0x1a9ea34)
4/8
VVFINgehstADJARoteNEMariaNNMannVAFINbistADVschon
HASH(0x1a9eb90)
schon
```

## Referenzen auf Subroutinen

```
$ergebnis = lines("Beispiel1","Test1");
$subref = \&lines;
$referenzergebnis = &{$subref}("Beispiel2","Test2");
print $ergebnis,"\n";
print $subref,"\n";
print $referenzergebnis,"\n";

sub lines {
    # @_ ist die Liste der übergebenen Parameter
    print "\n$_[0] $_[1] ", "*" x 40 , " $_[1]";
    return "xyz";
}

Ausgabe:
Beispiel1 Test1 ***** Test1
Beispiel2 Test2 *****
Test2xyz
CODE(0x1a72990)
xyz
```

## "mehrdimensionale DS"

```
$array[7][12]           # array of arrays
$array[7]{string}       # array of hashes
$hash[string][7]       # hash of arrays
$hash{string}{another string} # hash of hashes

Z.B.: Array of Array = Eine Liste, die als Elemente wieder Listen enthält

@AoA = ( [ "geh" ],
          [ "e", "st", "t" ],
          [ "en", "t", "en" ] );
```

## Arrays of arrays - Zugriff (1)

```
@AoA = ( [ "geh" ],
          [ "e", "st", "t" ],
          [ "en", "t", "en" ] );

print "ich ", $AoA[0][0].$AoA[1][0], "\n";
print "du ", $AoA[0][0].$AoA[1][1], "\n";
print "er/sie/es ", $AoA[0][0].$AoA[1][2], "\n";
print "wir ", $AoA[0][0].$AoA[2][0], "\n";
print "ihr ", $AoA[0][0].$AoA[2][1], "\n";
print "sie ", $AoA[0][0].$AoA[2][2], "\n";

$Stamm = $AoA[0][0];
$perlsg = $AoA[1][0];
$per2sg = $AoA[1][1];

print "Ich ".$Stamm.$perlsg." auf die TaCoS, ";
print $Stamm.$per2sg." Du mit?\n";
```

## Arrays of arrays - Zugriff (2)

```
@paradigma = ( [ "geh", "kling", "ring", "ruf",
                "sing", "schwimm", "trink" ],
               [ "e", "st", "t" ],
               [ "en", "t", "en" ],
               [ "ging", "klang", "rang", "rief",
                 "sang", "schwamm", "trank" ],
               [ "", "st", "" ],
               [ "en", "t", "en" ] );

# Paradigma der Praesensformen ausgeben
for $h ( 0 .. $# { @paradigma[0] } ) {
    print "Stamm: $paradigma[0] [$h]\n";
    for $i ( 1 .. $# { @paradigma[1] } ) {
        for $j ( 0 .. $# { $paradigma[$i] } ) {
            print $paradigma[0] [$h].$paradigma[$i] [$j], "\n";
        }
    }
}
```

## Hashes - (1)

- Hashes (assoziative Listen) werden in anderen Programmiersprachen oftmals auch als **"Dictionaries"** bezeichnet.
- Sie werden mit dem Zeichen **%** eingeleitet.
- Statt mit Indices versehenen Werten haben Sie Schlüssel (**keys**) mit dazugehörigen Werten (**values**).
- Der Übersichtlichkeit halber kann man statt dem Komma Schlüssel und Werte mit **"=>"** voneinander trennen.

```
%vform = (Stamm => 'geh',
          Person => 1,
          Numerus => 'Sg',
          Modus => 'Indikativ',
          GV => 'Aktiv', # Genus Verbi
          Endung => 'e',
          VForm => 'gehe');
```

## Hashes - (2)

```
%vform = ('Stamm', 'geh',
          'Person', 1,
          'Numerus', 'Sg',
          'Modus', 'Indikativ',
          'GV', 'Aktiv',
          'Endung', 'e',
          'VForm', 'gehe');
```

**entspricht...**

```
%vform = ('Stamm' => 'geh',
          'Person' => 1,
          'Numerus' => 'Sg',
          'Modus' => 'Indikativ',
          'GV' => 'Aktiv',
          'Endung' => 'e',
          'VForm' => 'gehe');
```

## Hashes - (3)

```
%xyz = ("Pu der Baer"      => "Saeugetier",
        "Der blaue Klaus" => "Ausserirdischer",
        "Heino der Saenger" => "Mensch",
        "Der Raecher der Enterbten" => "Mensch",
    );
```

Keys können auch aus mehr als einem Wort bestehen, müssen dann aber in doppelten Anführungszeichen stehen.

## Hashes - Zugriff (1)

```
#!/usr/bin/perl -w

%vform = ('Stamm' => 'geh',
          'Person' => 1,
          'Numerus' => 'Sg',
          'Modus' => 'Indikativ',
          'GV' => 'Aktiv', # Genus Verbi
          'Endung' => 'e',
          'VForm' => 'gehe');

print $vform{Stamm};
print $vform{"Stamm"};
```

## Hashes - Zugriff (2)

```
#!/usr/bin/perl -w

%vform = ('Stamm' => 'geh',
          'Person' => 1,
          'Numerus' => 'Sg',
          'Modus' => 'Indikativ',
          'GV' => 'Aktiv', # Genus Verbi
          'Endung' => 'e',
          'VForm' => 'gehe');

my @array_of_keys = keys(%vform);

foreach(@array_of_keys) {
    print $_."\n";
}
```

## Hashes - Funktion keys

```
#!/usr/bin/perl -w

%vform = ('Stamm' => 'geh',
          'Person' => 1,
          'Numerus' => 'Sg',
          'Modus' => 'Indikativ',
          'GV' => 'Aktiv', # Genus Verbi
          'Endung' => 'e',
          'VForm' => 'gehe');

@vform = keys %vform;
print @vform;
# @vform enthält ( 'Stamm', 'Person', ... , 'VForm' )

Hashes sind ungeordnet, was sich an der willkürlichen Reihenfolge der Schlüssel, die nun in @vform gespeichert sind, erkennen läßt.
```

## Hashes - Funktion values

```
#!/usr/bin/perl -w

%vform = ('Stamm' => 'geh',
          'Person' => 1,
          'Numerus' => 'Sg',
          'Modus' => 'Indikativ',
          'GV' => 'Aktiv', # Genus Verbi
          'Endung' => 'e',
          'VForm' => 'gehe');

my @array_of_values = values(%vform);
foreach(@array_of_values) {
    print $_."\n";
}
```

## Hashes of Arrays

```
# NN = Normales Nomen (STTS)

%tag = ("Kategorie" => ["NN"],
        "Genus" => ["Masc", "Fem", "Neut"],
        "Kasus" => ["Nom", "Gen", "Dat", "Akk"],
        "Numerus" => ["Sg", "Pl"],
        "Flexion" => ["Sw", "St", "Mix"],
    );

foreach my $i (0 .. $#{$tag{"Kasus"}}) {
    print $tag{"Kasus"}[$i]."\n";
}
```

## Hashes of Arrays - Zugriff (1)

```
# NN = Normales Nomen (STTS)

%tag = ( "Kategorie" => ["NN"],
        "Genus"     => ["Masc", "Fem", "Neut"],
        "Kasus"    => ["Nom", "Gen", "Dat", "Akk"],
        "Numerus"  => ["Sg", "Pl"],
        "Flexion"  => ["Sw", "St", "Mix"],
    );

print $tag{Kasus}[0];      # Nom
print $tag{Kasus}[1];      # Gen
print $tag{Kasus}[2];      # Dat
print $tag{Kasus}[3];      # Akk
```

## Hashes of Arrays - Zugriff (2)

```
# NN = Normales Nomen (STTS)

%tag = ( "Kategorie" => ["NN"],
        "Genus"     => ["Masc", "Fem", "Neut"],
        "Kasus"    => ["Nom", "Gen", "Dat", "Akk"],
        "Numerus"  => ["Sg", "Pl"],
        "Flexion"  => ["Sw", "St", "Mix"],
    );

foreach my $i (0 .. $#{$tag{"Kasus"}}) {
    print $tag{"Kasus"}[$i]."\n";
}
```

### Ausgabe:

Nom  
Gen  
Dat  
Akk

## Hashes of Arrays - Zugriff (3)

```
foreach $feature (keys %tag) {
    print "$feature: @{$tag{$feature}}\n"
}
```

### AUSGABE:

Flexion: Sw St Mix  
Kasus: Nom Gen Dat Akk  
Kategorie: NN  
Genus: Masc Fem Neut  
Numerus: Sg Pl

## Hashes of Arrays - Zugriff (4)

```
my @array_of_keys = keys(%vform);
foreach(@array_of_keys) {
    print $_."\n";
}
```

### AUSGABE:

GV  
Endung  
Person  
Stamm  
VForm  
Modus  
Numerus

## Hashes of Arrays - Zugriff (5)

```
my @array_of_values = values(%vform);
foreach(@array_of_values) {
    print $_."\n";
}
```

### AUSGABE:

Aktiv  
e  
1  
geh  
gehe  
Indikativ  
Sg

## Arrays of hashes

```
@paradigma = ( { Stamm => 'geh',
                Person => 1,
                Numerus => 'Sg',
                Endung => 'e',
                VForm => 'gehe' },
              { Stamm => 'geh',
                Person => 2,
                Numerus => 'Sg',
                Endung => 'st',
                VForm => 'gehst' },
              { Stamm => 'geh',
                Person => 3,
                Numerus => 'Sg',
                Endung => 't',
                VForm => 'geht' },
    );
```

## Arrays of hashes - Zugriff (1)

```
print $paradigma[0]{VForm};
print $paradigma[1]{VForm};
```

### Ausgabe:

```
gehegeht
```

## Arrays of hashes - Zugriff (2)

```
# prints the whole thing with refs
for $href ( @paradigma ) {
    print "{ ";
    for $role ( keys %$href ) {
        print "$role=$href->{$role} ";
    }
    print "}\n";
}
```

### Ausgabe:

```
{ Endung=e Person=1 Stamm=geh VForm=gehe Numerus=Sg }
{ Endung=st Person=2 Stamm=geh VForm=gehst Numerus=Sg }
{ Endung=t Person=3 Stamm=geh VForm=geht Numerus=Sg }
```

## Arrays of hashes - Zugriff (3)

```
# print the whole thing with indices
for $i ( 0 .. $#paradigma ) {
    print "$i is { ";
    for $role ( keys %{ $paradigma[$i] } ) {
        print "$role=$paradigma[$i]{$role} ";
    }
    print "}\n";
}
```

### Ausgabe:

```
0 is { Endung=e Person=1 Stamm=geh VForm=gehe Numerus=Sg }
1 is { Endung=st Person=2 Stamm=geh VForm=gehst Numerus=Sg }
2 is { Endung=t Person=3 Stamm=geh VForm=geht Numerus=Sg }
```

## Arrays of hashes - Zugriff (4)

```
# print the whole thing one at a time
for $i ( 0 .. $#paradigma ) {
    for $role ( keys %{ $paradigma[$i] } ) {
        print "Index: $i Key: $role Value:
$paradigma[$i]{$role}\n";
    }
}
```

### Ausgabe:

```
Index: 0 Key: Endung Value: e
Index: 0 Key: Person Value: 1
Index: 0 Key: Stamm Value: geh
Index: 0 Key: VForm Value: gehe
Index: 0 Key: Numerus Value: Sg
Index: 1 Key: Endung Value: st
Index: 1 Key: Person Value: 2
usw.
```

## Hashes of Hashes

```
%HoH = ( flintstones => { lead => "fred",
                        pal   => "barney", },
         jetsons    => { lead => "george",
                        wife  => "jane",
                        "his boy" => "elroy", },
         simpsons   => { lead => "homer",
                        wife  => "marge",
                        kid   => "bart", },
);
```

## Hashes of Hashes -Zugriff (1)

```
%HoH = ( flintstones => { lead => "fred",
                        pal   => "barney", },
         jetsons    => { lead => "george",
                        wife  => "jane",
                        "his boy" => "elroy", },
         simpsons   => { lead => "homer",
                        wife  => "marge",
                        kid   => "bart", },
);
```

```
print $HoH{flintstones}{lead};
```

```
$HoH{flintstones}{wife} = "wilma";
print $HoH{flintstones}{wife};
```

### Ausgabe:

```
fredwilma
```

## Hashes of Hashes -Zugriff (2)

```
# print the whole thing
foreach my $family ( keys %HoH ) {
    print "$family: { ";
    for $role ( keys %{ $HoH{$family} } ) {
        print "$role=$HoH{$family}{$role} ";
    }
    print "}\n";
}
```

### Ausgabe:

```
simpsons: { kid=bart lead=homer wife=marge }
jetsons: { lead=george wife=jane his boy=elroy }
flintstones: { lead=fred wifewilma pal=barney }
```

## Vordefinierte Variablen - \$\_

`$_` enthält den jeweils aktuellen Wert bzw. das jeweils aktuelle Argument

```
$string = 'Hallo';
chop($string);
print $string."\n";
```

```
$_ = 'Hallo';
chop($_);
print $_, "\n";
```

```
$_ = 'Hallo';
chop;
print;
```

### Ausgabe:

```
Hall
Hall
Hall
```

## Vordefinierte Variablen - @\_

`@_` enthält die Parameter, die beim Aufruf einer Subroutine übergeben wurden (siehe "Subroutinen")

```
sub lines {
    # @_ ist die Liste der übergebenen Parameter
    print "\n$_[0] $_[1] ", "*" x 40, " $_[1]\n";
}
```

Mit `$_[0]`, `$_[1]`, `$_[2]` etc. sind der erste, zweite, dritte etc. Parameter ansprechbar.

## Vordefinierte Variablen - @ARGV

`@ARGV` enthält die Parameter, die beim Aufruf des Perl-Skripts mit übergeben wurden

```
$ARGV[0] or die "\nAufruf: sort.pl DATEI\n";
```

Das Perl-Skript braucht einen Parameter beim Aufruf, bekommt es den nicht, wird eine Bemerkung ausgegeben und danach abgebrochen.

## Funktionen für Zeichenketten

<code>chomp</code>	letztes Zeichen entfernen, sofern Separator
<code>chop</code>	letztes Zeichen entfernen
<code>chr</code>	Zeichen eines Zeichenwerts ermitteln
<code>crypt</code>	Zeichenkette verschlüsseln
<code>index</code>	erstes Vorkommen einer Teilzeichenkette in Zeichenkette suchen
<code>lc</code>	alle Zeichen einer Zeichenkette in Kleinbuchstaben umwandeln
<code>lcfirst</code>	erstes Zeichen einer Zeichenkette in Kleinbuchstabe umwandeln
<code>length</code>	Anzahl Zeichen einer Zeichenkette ermitteln
<code>ord</code>	Zeichenwert eines Zeichens ermitteln
<code>pack</code>	Binärdaten erzeugen
<code>pos</code>	Position der Anwendung von <code>m/[regexp]/g</code> auf Zeichenkette ermitteln
<code>reverse</code>	Zeichenkette von hinten nach vorn umwandeln
<code>split</code>	Zeichenkette in mehrere Zeichenketten aufsplitten
<code>substr</code>	Teilzeichenkette aus Zeichenkette extrahieren
<code>uc</code>	alle Zeichen einer Zeichenkette in Großbuchstaben umwandeln
<code>ucfirst</code>	erstes Zeichen einer Zeichenkette in Großbuchstabe umwandeln
<code>unpack</code>	Binärdaten auflösen

<http://selfhtml.teamone.de>

## Funktionen für Arrays und Hashes

<code>delete</code>	Elementpaar aus Hash löschen
<code>each</code>	nächstes Elementpaar aus Hash ermitteln
<code>exists</code>	Ermitteln ob ein Hash-Namen existiert
<code>grep</code>	Teilliste aus Liste durch Bedingung extrahieren
<code>join</code>	Liste in Zeichenkette verwandeln
<code>keys</code>	Alle Namen eines Hashes ermitteln
<code>map</code>	Befehle auf alle Listenelemente anwenden
<code>pop</code>	letztes Element eines Arrays löschen
<code>push</code>	Elemente an einen Array anhängen
<code>reverse</code>	Reihenfolge der Listenelemente umkehren
<code>shift</code>	erstes Element eines Arrays löschen
<code>sort</code>	Listenelemente sortieren
<code>splice</code>	Elemente innerhalb eines Arrays löschen, hinzufügen, ersetzen
<code>undef</code>	Wert aus Hash oder Array entfernen
<code>unshift</code>	Elemente am Anfang eines Arrays hinzufügen
<code>values</code>	Alle Werte eines Hashes ermitteln

<http://selfhtml.teamone.de>

## Subroutinen (1)

```
#!/usr/bin/perl -w

print "Hallo Perl.\n";
&hallo;
hallo(); # Alternativ

sub hallo {
    print "Hallo Welt.\n";
}
```

- Subroutinen lagern Programmteile aus.
- Diese können mit einem einzigen Aufruf an mehreren Stellen aufgerufen werden.
- Der Aufruf erfolgt mit dem Namen des Unterprogramms und einem vorangestellten `&`, oder alternativ ohne `&`, dafür aber mit nachgestellten Klammern
- Eine Subroutine kann an beliebiger Stelle im Programm definiert werden.

## Subroutinen (2)

```
#!/usr/bin/perl -w

sub hallo;

print "Hallo Perl\n";
hallo;

sub hallo {
    print "Hallo Welt\n";
}
```

- Manchmal genügt auch allein der Name, allerdings nur wie im obigen Beispiel.

## Subroutinen - Parameter (1)

```
sub lines {
    local($i);
    print "\n\n@_ ". "*" x 50 , " @_ \n\n" ;
}
```

### Aufruf z.B. mit:

```
lines(4);
```

### Ausgabe:

```
4 ***** 4
```

- In dem **vordefinierten Array** `@_` sind die Parameter gespeichert, hier nur der Wert 4

## Subroutinen - Parameter (2)

```
sub lines {
    local($i);
    print "\n\n$_[0] $_[1] ". "*" x 50 , " $_[1] \n\n" ;
}
```

### Aufruf z.B. mit:

```
lines("Beispiel",4);
```

### Ausgabe:

```
Beispiel 4 ***** 4
```

- In dem **vordefinierten Array** `@_` sind die Parameter gespeichert, hier die Werte "Beispiel" und 4. Der Zugriff erfolgt mit der vordefinierten **Skalar** `$_` und dem entsprechenden nachgestellten Index.

## URL

Das alles und noch viel mehr...

- <http://www.perl.com>
- <http://www.perldoc.com/>
- <http://www.perl.org>
- <http://www.perl.de>
- <http://www.perlmonks.org>
- <http://selfhtml.teamone.de>
- <http://www.ora.com/catalog/ppperl2/noframes.html>
- <http://www.u.arizona.edu/~hammond/>
- <http://www.sysadminmag.com/tpj/>