Skriptsprachenorientierte Programmiertechnik

Perl Teil 7

Prof. Ilse Hartmann

Gliederung

Funktion index

- **Suche Substring in String: index**
  Bsp: **$where = index ($big, $small);**

  liefert erste Position von $small in $big bzw. -1, wenn nicht gefunden

- **Suche Substring in String ab bestimmter Position**

  Bsp.: **$where1 = index ($big, $small);**
  $where2 = index ($big, $small, $where+1);

  Dritter Parameter: minimaler Wert für return-Value

- **rindex**: liefert Index des letzten Vorkommens
  Bsp.: **my $lastSlash = rindex(“/etc/passwd“,“/“);** #liefert 4

Funktion substr

- **Manipulation eines Teilstrings: substr**
  - + bestimmt aus einem Skalar einen Teilstring ab einer angegebenen Stelle und liefert diesen zurück.
  - + Man kann angeben, wie lange der Teil sein soll.
  - + Zusätzlich kann der Ersatzwert angegeben werden, der an Stelle des Teiles eingefügt wird.
  - + Dem Rückgabewert der Funktion kann auch ein Wert zugewiesen werden.

- **Beispiele:**

```
my $text = '123456';
print substr($text,2);        #3456
print substr($text,2,2,'#');  # 34
print "\n$text\n";            #12#56
```

Einfache Ersetzung einzelner Zeichen ohne reguläre Ausdrücke

tr-Syntax (*translate):* ***tr/suchzeichen/ersetzungen/***

- Auch Zeichenbereiche können per Bindestrich (-) angegeben werden
- Auf der rechten Seite müssen (mindestens) ebenso viele Zeichen stehen wie links!

```
$_ = "chinesisches Roulette rigoros.";
tr/Rr/Ll/; # Ersetze alle R und r durch L bzw. l
print "$_\n";
my $klein = "alles in kleinbuchstaben.";
$klein =~ tr/a-z/A-Z/; # Ersetzen von Bereichen (hier a bis z)
print "$klein\n";
my $test = "abcde";
$test =~ tr/abcd/AB/; # Zuwenig rechts: B ersetzt auch c und d!
print "$test\n"; # „ABBBe“
```

Weitere Kontrollstrukturen

Kategorie	Schlüsselwörter
Umkehrung von Bedingungen	**unless** **until**
Statement-Modification (Nachgestellte Bedingungen)	**... if (...)** **... while (...)** **... foreach (...)**
Schleifensteuerung	**last** **next** **redo**

unless

- **Negatives Pendant zu if**
- **unless entspricht if (!...), d. h. der Codeblock wird ausgeführt, wenn die Bedingung nicht wahr ist**
- **Ebenfalls gleichbedeutend: Verlagerung in den else-Teil**

```
unless ($zahl >= 0)
{
        print "Die Zahl ist negativ!\n";
}
if (!($zahl >= 0))
{
        print "Die Zahl ist negativ!\n";
}
if ($zahl >= 0) {
# Nichts...
}
else {
        print "Die Zahl ist negativ!\n";
}
```

until

- Negatives Pendant zu while
- until entspricht while (!...), d. h. der Codeblock wird als Schleife ausgeführt, bis die Bedingung wahr ist
- Wie while ist auch until kopfgesteuert
- Fußgesteuerte Variante: do/until

```
$zaehler = 0;
until ($zaehler > 10)
{
$zaehler += 2;
            print "Der Zähler steht jetzt auf $zaehler.\n";
}
# Ausgabe: 2, 4, 6, 8, 10 und 12
# Fußgesteuert:
do
{
...
}
until (...)
```

Statement Modification

- **Kompaktere Schreibweise für if, unless, while, until und foreach**
- **Anweisung steht vorne, die Bedingung hinten**
- **Bedingungsteil wird dennoch zuerst ausgewertet!**
- **Klammern um die Bedingung können hier weggelassen werden**
- **Nur möglich bei einzelner Anweisung (keine Blöcke)**
- **Bei foreach ist nur $_ als Kontrollvariable zulässig**

Statement Modification

Beispiel:

```
print "$n ist eine negative Zahl.\n" if $n < 0;

&fehler("Falsche Eingabe") unless &gueltig($eingabe);

print " " . ($i++) . "\n“
while $i < 10;

$i *= 2 until $i > $j;

&gruessen($_) foreach @person;
```

Last

- **Beendet (verläßt) eine Schleife sofort**
- **Entspricht dem break-Operator in C**
- **Bei verschachtelten Blöcken wird nur aus dem innersten Block gesprungen**

```
while (1) # Endlosschleife!
{
        chomp (my $eingabe = <STDIN>);
        if ($eingabe eq '')
        {
                last; # „Notausgang“
        }
        print "Sie haben eingegeben: $eingabe\n";
}
print "Die Schleife ist beendet.\n";
```

Next

- Sprung zum Schleifenende, d.h. Abbruch des aktuellen Durchlaufes
- Nächster Schleifendurchlauf wird ausgeführt, sofern die Bedingung erfüllt ist
- Entspricht dem continue-Operator in C

```
my $anzahl = 0;
my %wortanzahl;
while (<STDIN>) # Aus Standardeingabe bis Strg+Z lesen
{
        foreach (split) # Jede Zeile an Whitespaces trennen, Worte
durchlaufen
        {
                if (/\W/) # Kommt ein Sonderzeichen vor?
                {
                        next; # Dann zum nächsten Wort springen
                }
                $anzahl++;
                $wortanzahl{$_}++;
        }
}
```

Redo

- **Sprung zum Schleifenbeginn, d.h. erneute Ausführung desselben Durchlaufes**
- **Keine erneute Prüfung der Schleifenbedingung**

```
if (($a < 0) || ($b < 0)) {
        print "Mindestens eine der Zahlen ist negativ.\n";
}
if (($n != 0) && ($summe/$n < 5)) {
        print "Der Durchschnittswert liegt unter 5.\n";
}
if (($a < 0) ^ ($b < 0)) {
        print "Genau eine der Zahlen ist negativ, nicht beide.\n";
}
```

Logische Operatoren

- **Operatoren für logisches or, and, not und xor**
- **Nutzbar als Zeichen (||, &&, ! und ^) oder als Wort (s.o.)**
- **|| und && sind sogenannte „Kurzschlußoperatoren“ (*short-circuit operators),***

```
if (($a < 0) || ($b < 0)) {
        print "Mindestens eine der Zahlen ist negativ.\n";
}
if (($n != 0) && ($summe/$n < 5)) {
        print "Der Durchschnittswert liegt unter 5.\n";
}
if (($a < 0) ^ ($b < 0)) {
        print "Genau eine der Zahlen ist negativ, nicht beide.\n";
}
```

Kurzschlussoperatoren

- **Reicht zur Auswertung eines logischen or bzw. and die linke Seite aus, so wird die rechte Seite nicht ausgewertet**
- **Vor allem gebräuchlich für die die-Funktion („öffne oder stirb!“)**

```
if (($n != 0) && ($summe/$n < 5)) # Division durch Null verhindern
{
        print "Der Durchschnittswert liegt unter 5.\n";
}
open (DATEI, $filename) or die "Kann '$filename' nicht öffnen: $!";
```