Der Flip-Flop-Operator

Hinter dem Flip-Flop-Operator versteckt sich der Range-Operator. Dessen gebräuchlichste Verwendung ist es wohl, Listen zu erstellen. In vielen Programmen sieht man so etwas wie

```
for( 1..10 ){
   # do anything
}
```

Hier wird eine Liste von 1 bis 10 aufgebaut. Mit dieser Eigenschaft dürften wohl die meisten den Range-Operator kennen. Weit seltener kommt der Range-Operator mit seiner Eigenschaft als Flip-Flop vor.

Im Skalaren Kontext liefert der Range-Operator einen Boolschen Wert zurück und ist somit „bistabil“. Diese Flip-Flop-Eigenschaft eignet sich zum Beispiel sehr gut, wenn man einen bestimmten Bereich aus einem Text ausgeben will und man nicht selbständig mit if's einen Boolschen Zustand pflegen will.

In Listing 1 ist ein Textausschnitt zu sehen, von dem nur der Text zwischen `START` und `STOP` interessant ist.

```
Dies ist ein längerer Text
mit vielen unnötigen Zeilen
vor dem eigentlich Wichtigen.

START
Wichtiger Text
über drei
Zeilen
STOP

Und noch unwichtigeren Zeilen
nachher
```

Listing 1

Listing 2 zeigt einen Code, wie man es mit einem eigenen „Flip-Flop“ machen kann (Der Text aus Listing 1 ist im `__DATA__`-Bereich):

```
#!/usr/bin/perl

use strict;
use warnings;

my $bool = 0;

while( my $line = <DATA> ){
   if( $line =~ /^START/ ){
      $bool = 1;
   }

   print $line if $bool;

   if( $line =~ /^STOP/ ){
      $bool = 0;
   }
}
```

Listing 2

Der Flip-Flop ist solange `false` wie der Ausdruck auf der linken Seite falsch ist. Ist der linke Ausdruck wahr, wird der Flip-Flop auch `true` und bleibt die solange bis der Ausdruck auf der rechten Seite wahr ist. Danach wird der Flip-Flop wieder `false`. Damit wird es recht einfach, den wichtigen Teil aus dem Text zu ziehen (siehe Listing 3).

```
#!/usr/bin/perl

use strict;
use warnings;

my $bool = 0;

while( my $line = <DATA> ){
   print $line if $line =
     ~ /^START/ .. $line =~ /^STOP/;
}
```

Listing 3

Wichtig ist noch zu wissen, dass jeder Flip-Flop sein eigenen Boolschen Status hat, so dass Flop-Flops auch verschachtelt werden können (siehe auch Listing 4).

```
if( $line =~ /^START/ .. $line =~ /^STOP/ ){
     if( $line =~ /ber/ .. $line =~ /drei/ ){
        chomp $line;
        print '*' . $line . "*\n";
     }
     else{
        print $line;
     }
  }
```

Listing 4

In diesem Listing ist aber auch ein anderes Phänomen dargestellt. Der Flip-Flop kann im selben Durchlauf erst `true` und gleich wieder `false` werden, der `if`-Block wird dennoch mindestens das eine Mal ausgeführt. Das hat damit zu tun, dass der rechte Ausdruck evaluiert wird, sobald der Range-Operator evaluiert wird. Dadurch ergibt sich die Ausgabe:

```
START
Wichtiger Text
*über drei*
Zeilen
STOP
```

Möchte man dieses Verhalten unterbinden, soll der rechte Ausdruck also erst im nächsten Durchlauf evaluiert werden, muss man statt `..` einfach `...` verwenden. Dadurch ergibt sich folgende Ausgabe:

```
START
Wichtiger Text
*über drei*
*Zeilen*
*STOP*
```

In den meisten Fällen liefern aber sowohl `...` als auch `..` die gleichen Ergebnisse.

Renée Bäcker

TPF-TICKER

Google Summer of Code 2008 -> Projekte der TPF

Beim diesjährigen „Google Summer of Code“ ist nach einer Pause auch die Perl-Foundation wieder vertreten.

Der Perl-Foundation wurden 6 Slots zugeteilt:

* Flesh out the Perl 6 Test Suite
* wxCPANPLUS
* Native Call Interface Signatures and Stubs Generation for Parrot
* Incremental Tricolor Garbage Collector
* Math::GSL
* Full Text Search Implementation for the content management system, Bricolage