Skriptsprachenorientierte Programmiertechnik

Perl Teil 3

Prof. Ilse Hartmann

Gliederung

Programmteile zum Aufruf aus anderen Stellen im Code

- Zur Auslagerung von häufig benutztem Code und/oder zur klaren Strukturierung

- Keine Unterscheidung zwischen „Funktion“ (liefert stets einen Rückgabewert) und „Prozedur“ (Routine ohne Rückgabewert)

- Eigener Namensraum, d.h. Routinennamen unterscheiden sich stets von Skalaren und Arrays

- Subroutinen können im Normalfall an jeder beliebigen Stelle im Quellcode stehen

- Definition einer Subroutine: **sub**

- Aufruf einer Subroutine: **Ampersand (&) und Name der Routine**

Beispiel Subroutine

Beispiel:

```
&refrain();
 print "Der Vater hüt' die Schaf'.\n";
print "Die Mutter schüttelt's Bäumelein,\n";
print "Da fällt herab ein Träumelein.\n";
&refrain();
sub refrain
{
      print "Schlaf, Kindlein schlaf!\n";
}
```

Parameter

- Auch ***Argumente*** *genannt*

- Werte, die an eine Subroutine übergeben werden

- Übergebene Parameterwerte werden in der speziellen Arrayvariable @_ gespeichert

- Erster Parameter ist also **$_[0]**, zweiter Parameter ist **$_[1]**, etc.

- **(Parameter sind lokal**, d.h. Veränderungen auf die Werte wirken sich nicht auf die originalen Werte aus)

Beispiel Parameter

Beispiel:

```
&refrain(2);
print "Der Vater hüt' die Schaf'.\n";
print "Die Mutter schüttelt's Bäumelein,\n";
print "Da fällt herab ein Träumelein.\n";
&refrain(3);
sub refrain
{
    foreach (1..$_[0])   # Refrainzeile in bestimmter Anzahl schreiben
    {
        print "Schlaf, Kindlein schlaf!\n";
    }
}
```

Rückgabewert

- Subroutinen geben immer einen Ergebniswert zurück, auch wenn dieser nicht genutzt wird

- Der zuletzt berechnete Wert einer Subroutine ist automatisch der Rückgabewert!

```
sub groessererWert # Subroutine kann auch am Anfang stehen
{
    if ($_[0] > $_[1])
    {
        $_[0]; # automatischer Rückgabewert
    }
    else {
        $_[1]; # automatischer Rückgabewert
    }
}
# Hier beginnt das Programm:
print &groessererWert(42, 13);
```

Rückgabewert (return)

- Der **return-Operator** beendet die Subroutine und liefert sofort einen Wert zurück
- **return ohne Wertangabe** liefert **undef** zurück

```
@namen = qw( Martha Hans Christian Fritz Beate );
$ergebnis = &getIndexOf("Fritz", @namen);
print "Ergebnis der Suche: $ergebnis.\n";
sub getIndexOf
{
    ($searchFor, @array) = @_;
    foreach (0..$#array) {
        if ($searchFor eq $array[$_]) {
            return $_;          # Treffer! Index sofort zurückgeben
        }
    }
    undef;      # Nicht gefunden. Rückgabe von undef
}
```

Nicht skalarer Rückgabewert

- Wird eine Subroutine im Listenkontext aufgerufen, liefert sie auch eine Liste als Ergebnis

- Ein return ohne Wert liefert in diesem Falle eine leere Liste (vgl. undef im skalaren Kontext)

```
@zahlen = &makeLotto();
print "Ihre Lottozahlen: @zahlen\n";
sub makeLotto
{
    foreach (0..5)
    {
        $lotto[$_] = 1 + int(rand(49));
    }
    @lotto;              # Rückgabe des Arrays
}
```

Variable Parameterlisten

Subroutinen definieren keinerlei Argumentlisten, d.h. der Aufruf ist mit beliebiger Art und Anzahl von Parametern möglich

- **Abfrage der Parameteranzahl ist per @_ in skalarem Kontext möglich**

```
sub groessererWert {
    if (@_ != 2) {
            print "Es müssen genau 2 Parameter übergeben werden!\n";
    }
            ...
}
```

- **Bessere Variante: Subroutine reagiert flexibel auf die Parameterliste:**

```
sub groessterWert {
    if (@_ != 0)
    {
            foreach (@_) {
            ...
            }
    }
}
```

Private Variable

- **Standardmäßig** sind alle Variablen eines Programms **globa**l, d.h. es kann von überall auf sie zugegriffen werden
- Der **my-Operator** legt private („lexikalische“) Variablen an
- **Geltungsbereich**: Nur innerhalb des Blocks

```
$a = 2;       # global!
$b = 3;       # global!
$c = &rechne($a, $b);

print "a=$a, b=$b, c=$c\n";
# a=8, b=6, c=8
sub rechne
{
        ($a, $b) = @_;
        # b verdoppeln und zu a addieren
        $a += ($b *= 2);
}
```

```
$a = 2; # global!
$b = 3; # global!
$c = &rechne($a, $b);

print “a=$a, b=$b, c=$c\n“;
#a= 2, b= 3,  c=8
sub rechne
{
        my ($a, $b) = @_;
        #b verdoppeln und zu a addieren
        $a += ($b *= 2);
}
```

Private Variable

- Lexikalische Variablen (**my**) können in jedem beliebigen Block deklariert werden (z.B. in if-Konstrukten oder Schleifen mit while oder for)

- **my** deklariert stets nur eine **einzelne Variable**

- mit my können auch private Arrays deklariert werden

```
$x = 42;            # Dieses $x ist global!
foreach my $x (1..5)          # $x ist nur für die Schleife gültig
{
        print "$x\n";
}
print "$x\n";

my $a, $b;          # Falsch! Nur $a wird deklariert!
my ($c, $d);        # Hier werden beide Skalare deklariert
my @namen;      # Deklaration eines privaten Arrays
```

Aufruf von Subroutinen

- **Das Ampersand-Zeichen (&, Kaufmanns-Und) kann zum Aufruf einer Subroutine weggelassen werden,**
  - + wenn die Subroutine vor ihrem Aufruf deklariert wird
  - + oder aus der Syntax klar erkennbar ist, daß es sich um einen Funktionsaufruf handelt (z.B. durch Klammern)
- **Das &-Zeichen muss jedoch immer benutzt werden, wenn es eine eingebaute Perl-Funktion gleichen Namens gibt**

```
sub sagHallo {
      print "Hallo!\n";
}
sagHallo; # Korrekt
```

```
sub sagHallo;    # Prototyp
sagHallo;          # Korrekt

sub sagHallo {
     print “Hallo!“;
}
```

```
sagHallo;        # FALSCH
sagHallo();     # Korrekt
sub sagHallo {
        print "Hallo!\n";
}
```

```
sub chomp {
    print "Mampf!\n";
}
chomp;         # FALSCH
chomp();      # FALSCH
&chomp;       # Korrekt
```

Aufruf der Perl-Funktion

Aufruf der eigenen Funktion

Zusammenfassung Aufruf

Prototyp	Bedeutung	Beispiel eines Aufrufs
sub NAME()	Keine Argumente	NAME();
sub NAME ($)	Ein skalares Argument	NAME(“Hans“);
sub NAME($$)	Zwei skalare Argumente	NAME (“Hans“,18),
sub NAME($$;$)	Zwei skalare Argumente und ein optionales Argument	NAME (“Hans“,18) NAME(“Hans“,18,‘m‘),
sub NAME(@)	Ein Feld oder eine Liste skalarer Argumente	NAME(@Hans); NAME (“Hans“,18,‘a‘,);
sub NAME ($@)	Ein skalares Argument und ein Feld oder eine Liste skalarer Argumente	NAME ($wert, @Feld); NAME(“Halle”,1,’a’,);
sub NAME (@$)	Unsinn!	