

## Übungsblatt #08

Bearbeitungszeitraum: bis Donnerstag, 17.12.2009

*Was will uns dieses Übungsblatt sagen?*

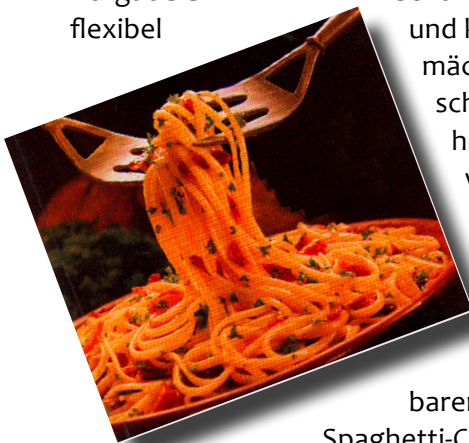
1. Komplexe Datenstrukturen lassen sich in Perl mit Hilfe von Referenzen (Zeigern) erreichen.
2. Somit sind Hashes von Hashes, Hashes von Listen, Listen von Listen, Listen von Hashes usw. möglich.
3. Aber auch Hashes von Hashes von Hashes oder Listen von Hashes von Listen usw. lassen sich problemlos realisieren.
4. Und auch eine Kombination verschiedener Datentypen ist denkbar: ein Hash kann sowohl Skalare, als auch Zeiger auf Listen, Hashes und Funktionen in „bunter Mischung“ enthalten. Somit lassen sich die aus der Sprache C bekannten „structs“ nachbilden.
5. Unter Verwendung derartiger Datenstrukturen kann ein Perl-Programm für sehr mächtige Datenabfrage- und Datenverwaltungs-Aufgaben verwendet werden.
6. Auch Zeiger auf Funktionen sind in Perl möglich.
7. Dies erlaubt insbesondere die Übergabe einer Funktion (bzw. einer Referenz auf diese Funktion) als Parameter.
8. Ohne einen im voraus durchdachten Programmierstil können längere Perl-Programme schnell unübersichtlich werden. Insofern ist es gerade bei Perl wichtig, sich vorab Gedanken über die Programmstruktur und vor allem über die Datenstrukturen zu machen.

### Eine Perl-basierte Mini-Datenbank

Um wirklich restlos jede(n) von Euch davon zu überzeugen, dass sich Perl auch für größere Datenverwaltungsaufgaben sehr gut eignet, wollen wir in diesem Übungsblatt einmal ein vollwertiges, interaktives Programm mit Perl schreiben und nicht lediglich „nur ein Skript“ wie bislang immer.

Die Aufgabe auf diesem Blatt lässt sich recht leicht mit Hilfe von Zeigern auf Listen, Zeigern auf Hashes und Zeigern auf Funktionen lösen. Wenn Ihr von diesen Konzepten Gebrauch macht, ist die Aufgabe ein flexibles

Gebrauch macht, ist die schöne Beispiel dafür, wie und kurz man in Perl sehr mächtige Programme schreiben kann. (Das heißt im Umkehrschluß: wenn Ihr nicht mit den genannten Konzepten arbeitet, kann Euer Programm auch schnell in nicht mehr wartbaren Code, sogenannten „Spaghetti-Code“ ausarten!)



Bevor Ihr loslegt solltet Ihr Euch deshalb ein paar grundlegende Gedanken zur Daten- und Programmstruktur machen. Am besten haltet Ihr Euch dazu an die Aufgabenstellung dieses Blattes.

### Worum geht's jetzt genau?

Auf unserer Homepage findet Ihr die Datei „access-Log.txt“, die einen Ausschnitt aus den Log-Dateien eines Web-Servers enthält. Hier werden neben IP-Adresse und Zeitstempel auch das verwendete Betriebssystem und die Dimension des Browser-Fensters registriert. (Diese Daten können vom Betreiber der Webseite ausgewertet werden, um das Design der Seiten so zu gestalten, dass es für die gängigsten Nutzungsprofile optimiert ist.)

Ziel dieses Übungsblatts ist es, ein vollwertiges Programm in Perl zu schreiben, das interaktiv (also im Dialog mit dem Benutzer) Auswertungen der Log-Daten erlaubt. In der Datei „beispielSession.txt“, die ebenfalls auf der Homepage steht, könnt Ihr einige Demo-Läufe des Programms in Ruhe nachvollziehen. Damit wird schnell klar, was das Programm leisten soll.

## Die Datenstruktur(en)

Noch vor dem Schreiben der ersten Zeile Perl-Code ist es ratsam, sich Gedanken über die Datenstruktur(en) zu machen, in der Ihr die Daten verwalten wollt. Wie wollt Ihr die Daten intern aufbewahren? Welche Datenstrukturen erscheinen Euch als geeignet? Wie sollen sie gefüllt werden? An welchen Stellen müsst Ihr wie auf die eingelesenen Informationen zugreifen (können)? Die Zugriffe sollen natürlich möglichst effizient funktionieren.

## Funktionen

Bislang haben wir immer nur Perl-Skripte geschrieben, die lediglich aus einem Hauptprogramm bestanden. Selbstverständlich lassen sich in Perl auch Funktionen verwenden - genau so, wie Ihr dies von C, Java (dort heißen sie *Methoden*) und anderen Programmiersprachen gewohnt seid.

Macht Euch deshalb Gedanken, welche Teile des Programms Ihr möglicherweise mehrfach verwenden wollt. Diese solltet Ihr dann - geeignet parametrisiert -

als Funktion zur Verfügung stellen. Mit Hilfe von Zeigern können die Funktionen dann bei Bedarf auch als einer anderen Funktion als Argumente übergeben werden. (Beispielsweise könnt Ihr einer Funktion, die eine Liste sortieren soll, die gewünschte Sortierfunktion als Zeiger übergeben.)

## Sortieren

Wir brauchen an verschiedenen Stellen verschiedene Sortierungen. Im Beispiel 1.46 im Skript habt Ihr gesehen, wie man der Sort-Funktion eine Funktion übergeben kann, die zum Sortieren verwendet wird. Überlegt Euch, wo es in diesem Fall für Euch Sinn machen wird, eigene Sortierungen zu beschreiben. (Tip: Ihr müsst ja auch nach IP-Adressen sortieren können. Diese korrekt zu sortieren ist ein wenig „tricky“.)

## Und los...

Nachdem nun die theoretischen Grundlagen vorliegen, könnt Ihr so langsam mit dem Programmieren beginnen. Wenn Eure vorab gemachten Gedanken sorgfältig waren, dann lässt sich mit wenig Aufwand das vollständige Programm nun schreiben.

### *Was soll das Programm denn jetzt konkret leisten?*

*Euer Programm soll beim Start zunächst die Daten einlesen, sie in geeigneten Strukturen ablegen und dann dem Benutzer das unten gezeigte Menü präsentieren. Abgesehen von Punkt 7 (Programm beenden), dienen alle Menüoptionen genau einer Abfrage der eingelesenen Daten. Dazu hat der Benutzer die Möglichkeit, einen regulären Ausdruck anzugeben, mit Hilfe dessen dann die IP-Adressen oder die Betriebssystem-Namen gefiltert werden. Ohne Angabe eines Filterkriteriums erhält er die Ausgabe aller vorhandenen Einträge.*

*Einen vollständigen Überblick über die Funktionsweise des Programms findet Ihr in der Datei „beispiel-session.txt“ auf unserer Homepage.*

Bitte wähle eine Funktion:

- 1) Vollständige Ausgabe der Zugriffe nach IPs gefiltert
- 2) Vollständige Ausgabe der Zugriffe nach Betriebssystem gefiltert
- 3) Anzahl der Zugriffe nach IPs (oder Teilen davon) ausgeben
- 4) Anzahl der Zugriffe nach Betriebssystem-Bezeichnung (oder Teilen davon) ausgeben
- 5) Hitliste der Zugriffe nach IPs (oder Teilen davon)
- 6) Hitliste der Zugriffe nach Betriebssystemen (oder Teilen davon)
- 7) Ende