Namespace, Pragmas, Module und Variablen

package MicroWeb;
Definiert den Namespace fuer alle folgenden Funktionen, Variablen und Anweisungen bis zur naechsten package-Anweisung zum Ende der Datei als „MicroWeb“.

our $foo;
Deklariert die Variable $foo im Namespace im aktuellen Scopes.

our $VERSION='0.7';
Deklariert die Variable $VERSION im Namespace im aktuellen Scopes und initialisiert sie mit dem skalaren Wert „0.7“.

my $socket;
Deklariert die Variable $socket im Namespace des aktuellen Scopes und reserviert einen Speicherort fuer diese.

use strict;
Laedt das Perl-Pragma „strict“. Dieses soll „schlampige“ Programmierung verbieten und damit viele typische Fehlerquellen ausmerzen.

use warnings;
Laedt das Perl-Pragma „warnings“, welches Warnmeldungen fuer das Auftreten von potenziellen Programmierfehlern ausgibt, z.B. die Auswertung nicht initialisierter Variablen, das Lesen von/Schreiben auf Dateihandler, die nicht geoeffnet sind, etc.

use IO::Socket::INET;
Laedt das Perl-Modul IO/Socket/INET.pm aus dem Bibliothekssuchpfad @INC.

Hashs

```
$conf={
        bind_address => '0.0.0.0',
    };
```
Generiert einen „anonymen Hash“ und speichert diesen in $conf, dieser wird bereits mit einem Eintrag gefuellt.
Hashs sind sogenannte assoziative Arrays, also Speicherkonstrukte mit 0 bis theoretisch unendlich vielen Eintraegen. Diese Eintraege bestehen stets aus 2 Elementen, einem Schluessel und einem Wert. Im Beispiel ist „bind_address“ der Schluessel und „0.0.0.0“ der Wert.

$host=$conf->{bind_address};
Liest den Wert hinter dem Schluessel „bind_address“ aus dem in $conf referenzierten Hash und speichert den Wert in $host.

Hashs koennen in Perl real oder referenziert sein. Ein „realer Hash“ liegt an Ort und Stelle im Speicher unter einem bestimmten Namen. Eine „Hash-Referenz“ ist ein Zeiger (Zeiger-Operator „->“) auf einen realen Hash.
Ein Spezialfall sind „anonyme Hashs“, wie hier im Beispiel. Diese werden im Speicher ohne einen Namen oder Scope angelegt und existieren nur dadurch, dass mindestens eine Referenz auf sie verweist.

Objekte

```
$socket=new IO::Socket::INET(
        Listen    => 5,
        Reuse     => 1
    );
```

Erzeugt ein neues Objekt der Klasse IO::Socket::INET unter Uebergabe der Hash-Parameter „Listen => 5“ und „Reuse => 1“. Das neue Objekt wird in $socket gespeichert.

$client=$socket->accept();
Rufe die Methode accept() des Objekts $socket auf und schreibe den Rueckgabewert in $client.

Schleifen und Bedingungen

Bedingungen sind Variablen oder feste Werte, die zu einem Wahr oder Falsch evaluiert werden. In Perl sind „0“, „“, 0 und undef aequivalent zu Falsch, alle anderen Eingaben werden mit Wahr bewertet.

```
while($bedingung){
	foo();
}
```
Solange die Bedingung $bedingung gilt, fuehre den Block mit foo(); aus.

next;
Ueberspringt den Rest einer Schleifeniteration, d.h. sofortiger Ruecksprung zum „while“ oder „for“ oder „foreach“ und fortsetzen mit der naechsten Iteration oder dem naechsten Element.

```
if($bedingung){
	foo();
}
```
Falls die Bedingung $bedingung gilt, fuehre den Block mit foo(); aus.

```
unless($bedingung){
	foo();
}
```
Falls die Bedingung $bedingung NIGHT gilt, fuehre den Block mit foo(); aus. Dies ist identisch mit if(not($bedingung))

next unless(defined($client));
Fuehrt den Befehl „next“ aus, wenn $client nicht undef ist.

Funktionen

```
sub processConnection($){
	my($client)=@_;
}
```
Erzeuge eine neue Funktion „processConnection“, die genau einen Parameter entgegen nimmt („($)“).
In dieser Funktion schreibe den ersten (und einzigen) Parameter in die neue Variable $client.

processConnection($client);
Rufe die Funktion „processConnection()“ auf und uebergebe ihr dabei den Wert von $client.

return(„bar“);
Beende die Funktion sofort und gebe den String „bar“ als Rueckgabewert zurueck.

$file=decodeUri($url);
Rufe die Funktion „decodeUri()“ auf und uebergebe ihr dabei den Wert von $uri.
Schreibe den Rueckgabewert der Funktion in $file.

Bedingungsoperatoren

(-d $file && $file=~/\/$/)
Bedingung, die wahr evaluiert, wenn der in $file gespeicherte Pfad ein Verzeichnis ist („-d“), und ob $file auf „/“ endet.

$a=($bedingung) ? ($rueckgabe_bei_true) : ($rueckgabe_bei_false);
Falls $bedingung zu Wahr evaluiert, schreibe den Inhalt von $rueckgabe_bei_true in $a – falls nicht, schreibe den Inhalt von $rueckgabe_bei_false in $a.

String-Operatoren

$file.=“bla“;
Haenge an den String in $file zusaetzlich den String „bla“ an. Dies ist aequivalent zu:
$file=$file.“bla“;

String-Operationen

($method,$url,$version)=split(/ /,$daten,3);
Teile den String in $daten in genau 3 Elemente, die durch Leerzeichen („/ /“) getrennt sind, und speichere sie in den Variablen $method, $url und $version.

$a=uc($method);
Verwandle alle kleingeschrieben ASCII-Zeichen in $method grossgeschriebene um und speichere das Ergebnis in $a.

Dateizugriff

$fh=new IO::File($file,'r');
Oeffne die Datei $file zum Lesen („r“) und schreibe das Handle der Datei in $fh.

binmode($fh);
Versetze das Dateihandle $fh in den Binaermodus.

read($fh,$buffer,4096);
Lese maximal 4096 Bytes von Handle $fh in $buffer.

print("Hello World!\n");
Schreibe „Hello World!“ mit Newline auf die Standardausgabe

print($client "HTTP/1.0 200 OK\r\n");
Schreibe „HTTP/1.0 200 OK“ und 2 Newlines in das Handle $client.

$line=readline($client);
Lese genau eine Zeile von Dateihandle $client und speichere diese in $line.

Programmsteuerung

die("Blah!\n");
Beende das Programm mit Exit-Code 1 und gebe die Meldung „Blah!“ mit Newline aus.

exit(0);
Beende das Programm mit Exit-Code 0.