Strukturen

Da das Übergeben von Werten an Funktionen mit Hilfe Zeigern doch sehr mühsam und fehleranfällig ist sind Strukturen erfunden worden. Mit Hilfe von Strukturen können Pakete von Variablen und Arrays geschnürt werden und diese dann in Funktionen aufgerufen werden.

Zum Beispiel.

```
#include <stdio.h>
struct waren
{
	char artikelbezeichnung[30];
	double nettopreis;
	double bruttopreis;
}artikel1, artikel2, artikel3;

struct waren artikel1 = {.artikelbezeichnung="Gummiente", .nettopreis=0.0, . nettopreis = 0.0};

struct waren artikel2 = {"Klopapier", 0.0, 0.0};

struct waren artikel3 = {0,0.0,0.0};

void eingabe ();
void bruttopreisf();

int main()
{
eingabe ();
bruttopreisf();

getchar();
return 0;
}

void eingabe()
{	printf("Geben Sie die Bezeichnung des 3. Artikels ein: ");
	scanf("%s",artikel3.artikelbezeichnung);
	while(getchar()!='\n');
	printf("\n");

	printf("Geben Sie den Nettopreis von %s ein: ", artikel1.artikelbezeichnung);
	scanf("%lf", &artikel1.nettopreis);
	while(getchar()!='\n');
	printf("\n");

	printf("Geben Sie den Nettopreis von %s ein:", artikel2.artikelbezeichnung);
	scanf("%lf", &artikel2.nettopreis);
	while(getchar()!='\n');
	printf("\n");

	printf("Geben Sie den Nettopreis von %s ein:", artikel3.artikelbezeichnung);
	scanf("%lf", &artikel3.nettopreis);
	while(getchar()!='\n');
	printf("\n");	}

void bruttopreisf()
{	artikel1.bruttopreis = artikel1.nettopreis * 1,19;
	printf("Der Bruttopreis von %s ist: %lf\n", artikel1. artikelbezeichnung,
	artikel1.bruttopreis);
	artikel2.bruttopreis = artikel2.nettopreis * 1,19;
	printf("Der Bruttopreis von %s ist: %lf\n", artikel2. artikelbezeichnung,
	artikel2.bruttopreis);
	artikel3.bruttopreis = artikel3.nettopreis * 1,19;
	printf("Der Bruttopreis von %s ist: %lf\n", artikel3. artikelbezeichnung,
	artikel3.bruttopreis);	}
```

Deklaration einer Struktur

Eine Struktur ist ein komplexer Datentyp, welcher aus einem Bündel von Variablen und Arrays zusammengesetzt ist.
Die Idee dahinter ist einfach zu verstehen: Da in einem Programm mitunter sehr viele Variable von einer Funktion an eine andere übergeben werden müssen, und das Auflisten der Variablen in den runden Klammern der Funktionen und auch im Funktionsaufruf sehr mühsam ist, haben sich findige Programmierer überlegt, daß es sehr viel einfacher wäre, diese Variablen und Arrays zu einem Paket zu packen und so als komplettes Paket von Funktion zu Funktion zu übergeben.
Genau das macht eine Struktur:

```
struct waren
{
	char artikelbezeichnung[30];
	double nettopreis;
	double bruttopreis;

}artikel1, artikel2, artikel3;
```

Hier wird ein **komplexer Datentyp** namens „waren“ erschaffen.
Ein **komplexer Datentyp** ist so etwas wie eine Vorlage, nach der dann beliebig viele Strukturen nach dieser Vorlage erstellt werden können

Das ist jetzt die Deklaration der Strukturen: Es werden drei Strukturen (artikel1, artikel2 und artikel3) deklariert. In jeder einezelnen Struktur gibt es einen char-Array „artikelbezeichnung“ und die beiden Variablen „nettopreis“ und „bruttopreis“.

Es gibt also drei Sätze von Variablen, daß heißt also drei Strukturen. Und deshalb gibt es auch den Array artikelbezeichnung drei Mal, die Variable nettopreis gibt es drei Mal und auch die Variable bruttopreis:

artikel1

```
char artikelbezeichnung[30];
double nettopreis;
double bruttopreis;
```

artikel2

```
char artikelbezeichnung[30];
double nettopreis;
double bruttopreis;
```

artikel3

```
char artikelbezeichnung[30];
double nettopreis;
double bruttopreis;
```

Es ist üblich Strukturen außerhalb der main und auch außerhalb aller anderer Funktionen zu deklarieren, da die Variablen global allen Funktionen zur Verfügung stehen sollen. Dies bedeutet aber auch, daß nur die Variablen und Arrays in eine Struktur gesteckt werde, die von mehreren Funktionen benötigt werden.

Initialisieren von Strukturen

! Achtung: in den geschweiften Klammern wird nur der komplexe Datentyp aus Elementardatentypen zusammengebastelt. Das ist nur der Plan für eine Struktur und nicht die Struktur selbst. Da aber die Struktur und damit die in ihr enthaltenen Variablen noch nicht existieren, können die Variablen und Arrays an dieser Stelle auch noch **nicht** initialisiert werden.

```
#include <stdio.h>
struct waren
{
        char artikelbezeichnung[30];
        double nettopreis;
        double bruttopreis;
}artikel1, artikel2, artikel3;
```

Hier werden die einzelnen Strukturen deklariert, das heißt dass die Strukturen und Variablen erst ab dieser Zeile existieren.

Deshalb erfolgt die Initialisierung aller Strukturen erst unter dem komplexen Datentyp. Ich habe hier mal drei verschiedene Möglichkeiten der Initialisierung dargestellt.

```
struct waren artikel1 = {.artikelbezeichnung="Gummiente", .nettopreis=0.0, . nettopreis = 0.0};
struct waren artikel2 = {"Klopapier", 0.0, 0.0};
struct waren artikel3 = {0,0.0,0.0};
```

Die Initialisierung muss außerhalb der main geschehen

```
void eingabe ();
void bruttopreisf();

int main()
{
eingabe ();
bruttopreisf();

getchar();
return 0;
}
```

Verwenden von Strukturen in Funktionen

Der Zugriff auf die einzelnen Variablen und Arrays der Struktur ist einfach:

Struktur.Variable

In unserem Fall wäre das z.B.

artikel1.nettopreis

Da es drei Strukturen gibt, gibt es auch drei Variable nettopreis. Damit klar wird, welcher der drei nettopreise gemeint ist, muß der Name der Struktur vorangestellt werden. Verbunden werden der Strukturname und die Variable durch einen Punkt.

Zum Beispiel

```
int main()
{
eingabe ();
bruttopreisf();
getchar();
return 0;
}
```

Im Übrigen gelten für die Verwendung von Stukturen die Regeln wie für Variable und Arrays auch:
artikel3.artikelbezeichnung ist ein Char-Array, deshalb wird bei der Verwendung in scanf kein & und auch keine eckigen Klammern benötigt.

```
void eingabe()
{     printf("Geben Sie die Bezeichnung des 3. Artikels ein: ");
      scanf("%s",artikel3.artikelbezeichnung);
      while(getchar()!='\n');
      printf("\n");
      printf("Geben Sie den Nettopreis von %s ein: ", artikel1.artikelbezeichnung);
      scanf("%lf", &artikel1.nettopreis);
      while(getchar()!='\n');
      printf("\n");
      printf("Geben Sie den Nettopreis von %s ein:", artikel2.artikelbezeichnung);
      scanf("%lf", &artikel2.nettopreis);
      while(getchar()!='\n');
      printf("\n");
      printf("Geben Sie den Nettopreis von %s ein:", artikel3.artikelbezeichnung);
      scanf("%lf", &artikel3.nettopreis);
      while(getchar()!='\n');
      printf("\n");      }
```

artikel3.nettopreis ist eine normale Variable, deshalb wird der Adressoperator & benötigt.

```
void bruttopreisf()
{     artikel1.bruttopreis = artikel1.nettopreis * 1,19;
      printf("Der Bruttopreis von %s ist: %lf\n", artikel1. artikelbezeichnung,
      artikel1.bruttopreis);
      artikel2.bruttopreis = artikel2.nettopreis * 1,19;
      printf("Der Bruttopreis von %s ist: %lf\n", artikel2. artikelbezeichnung,
      artikel2.bruttopreis);
      artikel3.bruttopreis = artikel3.nettopreis * 1,19;
      printf("Der Bruttopreis von %s ist: %lf\n", artikel3. artikelbezeichnung,
      artikel3.bruttopreis);    }
```