Neben den einfachen Datentypen **`char, int, float, …`** lassen sich komplexe Datentypen aufbauen, welche sich aus den einfachen Datentypen zusammensetzen.

Das soll am Beispiel einer **Tabelle** beschrieben werden.

Ein Versandhandel für Computerartikel möchte ihren Lagerbestand mit Hilfe der EDV verwalten. Dazu wird jeder Artikel mit einer eindeutigen (vierstelligen) Artikelnummer versehen. Gespeichert werden soll auch die Bezeichnung des Artikels, die vorhandene Stückzahl und der Einzelpreis. Wird die Tabelle durch die EDV-Abteilung des Unternehmens regelmäßig gepflegt, lässt sich jederzeit feststellen, ob und wie viele bestimmte Artikel vorhanden sind. Außerdem lassen sich beispielsweise der Lagerbestand ermitteln (für eine Inventur) oder eine automatische Rechnungsschreibung realisieren.

Aufbau der Tabelle:

Nr	*Name*	*Anzahl*	*Preis*
1000	Harddisk	10	45,00
1500	Drucker	4	95,00
1200	Monitor	6	245,00
2000	PC-Gehäuse	12	45,00
1600	CD-Rohlinge	100	9,95
…	…	…	…

Jede Spalte hat einen **Datenfeldbezeichner** (Nr, Name, Anzahl, Preis). Jede Zeile bildet einen **Datensatz** (engl. Record). Die einzelnen Datenfelder nennt man auch **Attribute** des Datensatzes. Die Gesamtheit aller Datensätze bilden die **Datei**. Jeder Eintrag in dieser Datei ist eindeutig durch seine Zeile und das Attribut festgelegt. Dateien oder Tabellen sind die Grundlagen aller **Datenbanken**.

Ein einzelner Datensatz in obigem Beispiel setzt sich zusammen aus vier grundlegenden Typen: Nr(int), Name(char*), Anzahl(int) und Preis(double).

Mit Hilfe der struct-Anweisung lässt sich der neue Datentyp beschreiben:

```
struct artikel  //Der neue Typ ist ‚artikel’
{
int nr;         //’nr’ ist vom Typ integer
char name[24];  //’name’ ist ein string mit maximal 24 Stellen
int anzahl;     //’anzahl’ ist vom Typ integer
double preis;   //’preis’ ist vom Typ double
};              //Achtung: struct-Anweisung unbedingt mit ‚;’ abschließen
```

Der Bezeichner ‚artikel’ ist jetzt ein neuer (benutzerdefinierter) Datentyp geworden.

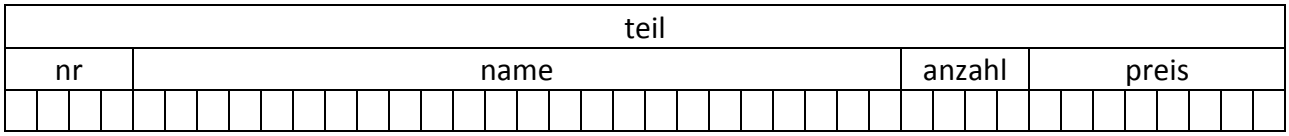
Mit der struct-Anweisung wird der neue Datentyp lediglich „beschrieben“. Will man einen Datensatz erfassen, so ist eine Deklaration zur Reservierung von Speicherplatz erforderlich:

```
artikel teil;   //’teil’ ist vom Typ artikel
```

Insgesamt wird ein Speicherbereich von 40 Byte reserviert (Begründung?).

Man nennt nr, name, anzahl und preis Elemente des Objektes teil.

Die Speicherbelegung wird durch folgende Abbildung dargestellt:



Um einzelne Elemente zu initialisieren erfolgt der Zugriff durch Angabe des Objektnamens in Verbindung mit dem Element. Objektname und Element sind durch einen ‚.' (Punkt) getrennt.

```
objektname.element    //Der ‚.'-Operator definiert das Element des Objektes.
```

In unserem Beispiel wären folgende Initialisierungen möglich:

```
teil.nr = 1000;
strcpy(teil.name,“Harddisk“);
teil.preis = 45.00;
. . .
```

Die Eingabe über Tastatur sähe beispielsweise folgendermaßen aus:

```
scanf_s(„%i“,&teil.nr);
scanf_s(“%lf”,&teil.preis);
flushall();                 //Eingabepuffer löschen; erforderlich!
gets_s(teil.name);          //’name’ ist ein Zeiger auf einen String
```

Bei der Eingabe des Strings ist eine Besonderheit zu beachten. Vor der Eingabe mit der gets_s()-Funktion müssen alle Eingabepuffer des Systems geleert werden. Das erfolgt mit der Funktion **`flushall()`**, welche vor **`gets_s()`** ausgeführt werden muss. Die Ursache liegt im System der Entwicklungsumgebung.

Analysieren und testen Sie das folgende Beispiel!

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

struct artikel        //Der neue Typ ist ‚artikel'
{                     //’artikel’ ist global in den folgenden Funktionen gültig
int nr;               //'nr' ist vom Typ integer
char name[24];        //'name' ist ein string mit maximal 24 Stellen
int anzahl;           //'anzahl' ist vom Typ integer
double preis;         //'preis' ist vom Typ double
};                    //Achtung: struct-Anweisung unbedingt mit ‚;' abschließen
```

```
int main()
{
      artikel teil;
      do
      {
      printf("\nDatensatz eingeben \n");
      printf("\nNummer: ");
      scanf_s("%i",&teil.nr);
      printf("\nName: ");
      _flushall();               //erforderlich! Löscht den Eingabepuffer
      gets_s(teil.name);
      printf("\nAnzahl: ");
      scanf_s("%i",&teil.anzahl);
      printf("\nPreis: ");
      scanf_s("%lf",&teil.preis);
      printf("\nLänge des Datensatzes: %i Zeichen\n",sizeof(teil));
      printf("\n%4i %20s %3i %8.2lf",teil.nr,teil.name,teil.anzahl,teil.preis);
      printf("\n\n'ESC' = Ende... ");
      }
      while(_getch()!=0x1b);
      return(0);
}
```

Ein Vorteil der Objekte, welche auf einer Struktur aufbauen, liegt darin, dass sich der ‚=' Zuweisungsoperator anwenden lässt, um Objekte zu kopieren.

Das Folgende Beispiel legt zwei Objekte **`teil1`** und **`teil2`** vom Typ **`artikel`** an. Eingegeben wird das Objekt **`teil1`**. Anschließend wird **`teil1`** in **`teil2`** kopiert und beide Datensätze ausgegeben.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct artikel
{
int nr;
char name[24];
int anzahl;
double preis;
};

int main()
{
      artikel teil1;
      artikel teil2;
      do
      {
      printf("\nDatensatz eingeben \n");
      printf("\nNummer: ");
      scanf_s("%i",&teil1.nr);
```

```
      printf("\nName: ");
      _flushall();              //erforderlich! Löscht den Eingabepuffer
      gets_s(teil1.name);
      printf("\nAnzahl: ");
      scanf_s("%i",&teil1.anzahl);
      printf("\nPreis: ");
      scanf_s("%lf",&teil1.preis);
      teil2=teil1;              //Objekt teil1 kopieren in Objekt teil2
      printf("\nDatensatz 1:\n");
      printf("\n%4i %20s %3i
%8.2lf",teil1.nr,teil1.name,teil1.anzahl,teil1.preis);
      printf("\nDatensatz 2:\n");
      printf("\n%4i %20s %3i
%8.2lf",teil2.nr,teil2.name,teil2.anzahl,teil2.preis);
      printf("\n\n'ESC' = Ende... ");
      }
      while(_getch()!=0x1b);
      return(0);
}
```

Übung:

Entwickeln Sie ein Programm, welches die Eingabe von zwei Datensätzen vom Typ **`artikel`** gestattet! Anschließend sollen die Inhalte beider Datensätze vertauscht und ausgegeben werden.