10. Strukturen

In **C** kann man als Programmierer **komplexe Datentypen,** die aus einfacheren Datentypen bestehen, als **Strukturen** definieren. Eine **Struktur** in **C** entspricht einer **einfachen Klasse** in **Java**, die **nur aus Attributen** besteht.

Def.1: (**C-Struktur**): Eine C-Struktur wird mit dem Schlüsselwort **struct** als Verbund von Attributen, deren Datentypen bereits bekannt sind, definiert. Dieser Verbund ist ein neuer Datentyp mit dem Datentypnamen STRNAME. STRNAME kann vom Programmierer als Bezeichner frei gewählt werden. Sind **a1, a2, …, aN** Attribute und **dt1, dt2, …, dtN** ihre Datentypen, dann wird eine C-Struktur STRNAME folgendermaßen definiert:

```
struct STRNAME
{dt1 a1;
 dt2 a2;
 …   …;
 dtN aN;
};
```

BSP.1: Eine C-Struktur für Artikel (Handelswaren):

```
struct Artikel
{int anr;
 char bez[21];
 double preis;
};
```

BSP.2: Eine C-Struktur für Mitarbeiter:

```
struct MitArb
{char name[21];
 double gehalt;
};
```

BSP.3: Eine C-Struktur für Anschriften:

```
struct Anschrift
{int plz;
 char ort[31];
 char strasspostf[41];
};
```

BSP.4: Eine C-Struktur für Mitarbeiter mit Anschriften:

```
struct MitArbAns
{struct MitArb m1;
 struct Anschrift a1;
};
```

Verwaltung von Variablen, deren Datentyp eine C-Struktur ist:

(V1) **Deklaration** und Speicherplatzbeschaffung für eine Variable **x**, deren Datentyp eine C-Struktur mit Strukturnamen STRNAME ist: **struct STRNAME x;**

BSP.5: Variablen vom Typ Artikel (s.o. BSP.1) werden deklariert und mit Speicherplatz angelegt: **struct Artikel w1, w3, w5[10];**

Auch Felder vom Datentyp einer C-Struktur können so angelegt werden.

(V2) **Zugriff** auf ein **Attribut a1** einer Strukturvariablen **x** mittels **Punktnotation** (wie in Java): **x.a1**

BSP.6:

```
struct Artikel x;
x.anr=4711;
```

```
strcpy(x.bez,”Seife”);
x.preis=1.98;
```

(V3) **C-Strukturen und Funktionen**
Eine Funktion kann ein Wert vom Typ einer C-Struktur zurückgeben:
BSP.7: Eine Funktion **einart()**, die den Tupelwert einer Artikel-Struktur (vgl. BSP.1) von der Tastatur einliest und zurückgibt:
Prototyp: **`Artikel einart(void);`**
Programm:

```
Artikel einart(void)
{Artikel z;
 printf("Artikelnummer     :");
 scanf("%d",&z.anr);
 printf("Artikelbezeichnung:");
 scanf("%s",z.bez);
 printf("Preis             :");
 scanf("%lf",&z.preis);
 return z;
}
```

Eine Funktion kann den Wert einer Strukturvariable übergeben bekommen:
BSP.8: Eine Funktion soll den Bruttopreis (Preis plus Mehrwertsteuer) eines Artikels berechnen und zurückgeben. Der Mehrwertsteuersatz wird auch übergeben:
Prototyp: **`double bruttopreis(struct Artikel w3, double mwst);`**
Programm:

```
double bruttopreis(struct Artikel w3, double mwst)
{double z=0.;
 z=w3.preis*(1.+mwst);
 return z;
}
```

Strukturdefinition mit dem Schlüsselwort `typedef`:
Um nicht immer das Wort **struct** in der Datentypbezeichnung **struct STRNAME** mitnehmen zu müssen, d.h. um nur die Datentypbezeichnung STRNAME verwenden zu können, kann man am Anfang des Quelltextes oder in einer Header-Datei die Definition einer C-Struktur mit **typedef** durchführen:

```
typedef struct
{dt1 a1;
 dt2 a2;
 …   …;
 dtN aN;
} STRNAME;
```

BSP.9: Definition einer Artikel-Struktur (vgl. BSP.1) mit **typedef**: Auszug aus einer Header-Datei **Artikel5.h** :

```
typedef struct
{int anr;
 char bez[21];
 double preis;
} Artikel5;
```