

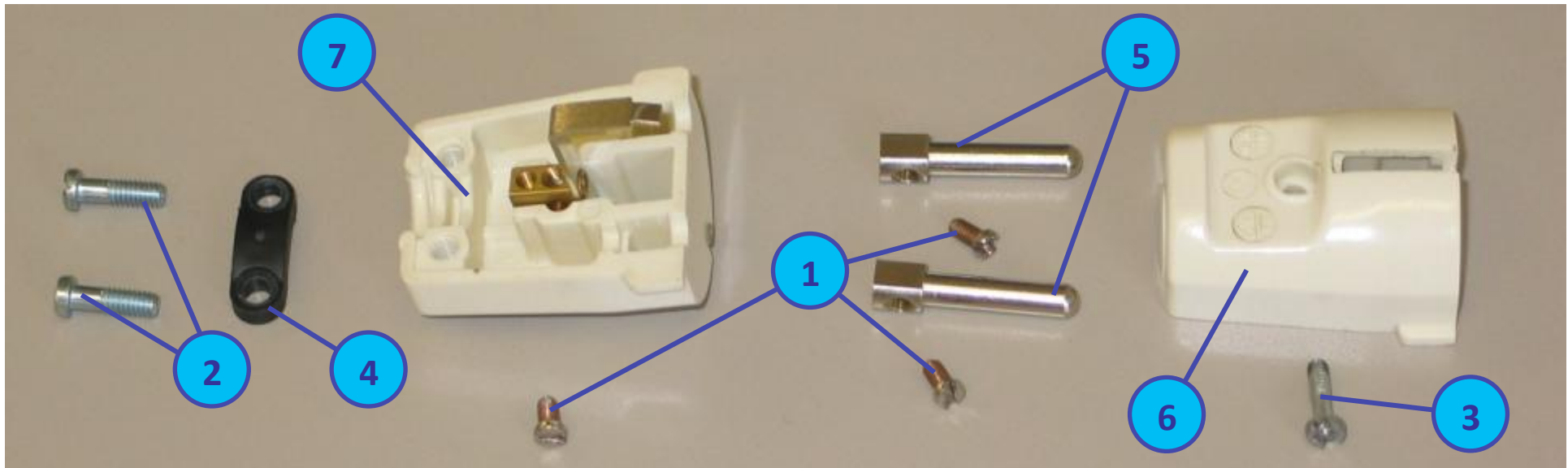


Hochschule für Angewandte  
Wissenschaften Hamburg  
*Hamburg University of Applied Sciences*

# Programmiertechniken 1

Strukturen

# Veranschaulichungsbeispiel Stückliste



Pos	Menge	Einheit	Benennung	Teilenummer
1	3	Stück	Kupferschraube	ST1001
2	2	Stück	Verbindungsschraube groß	ST1002
3	1	Stück	Verbindungsschraube klein	ST1003
4	1	Stück	Zugentlastung	ST1010
5	2	Stück	Einsteckverbindung	ST1011
6	1	Stück	Gehäusehälfte ohne Erdung	ST1020
7	1	Stück	Gehäusehälfte mit Erdung	ST1021

# Stückliste

## Stücklisten:

In der Gesamtzeichnung erhält jedes Werkstück eine Teilenummer, die sogenannte Positionsnummer. Die Positionsnummern, die möglichst dem Zusammenbau entsprechen sollen, stehen verbunden mit einer Hinweislinie neben der Darstellung. Außerdem sind die Positionsnummern in der Stückliste eingetragen. Aus der Stückliste der Zeichnung kann anhand der Positionsnummer die Menge, Einheit, Benennung, Norm-Kurzbezeichnung, und der Werkstoff des Einzelteils entnommen werden.

a) Beispiel für eine „klassische“ genormte Stückliste Form A nach DIN 6771-2:

Pos.	Menge	Einheit	Benennung	Sachnummer	Bemerkung
1	1	Stck	Antriebswelle	MT152.00.01.020	C 45
2	2	Stck	Antriebsrad	MT152.00.00.004	16 Mn Cr 5 BG
...	4	Stck	Sechskantschraube DIN 933–M8x25-8.8	NT 000.01.30.037	
9	1	Stck	Gehäuse	MT152.10.01.001	GG-20
...	...	...	...	...	...

b) Beispiel für die Kopfzeile einer heutigen „industriellen“ Stückliste (jede Firma hat eigenen Standard):

Item	Quantity	QC	Descriptions/Dimensions	Standard	Material	Mat. no.	Draw.No	F	DI/PI	Weight
							Erz Bg.		AC	

# Datenorganisation

Beispiel: Liste von Punkten im 3D-Raum

Index	X-Koordinate	Y-Koordinate	Z-Koordinate
0	23	36	-89
1	42	56	21
2	207	23	-23
3	-207	78	0
4	-64	99	0
5	-111	46	1

Array-Organisation

Mögliche Organisation:

- Drei **Arrays** mit den Koordinaten
- Ansprechen des Punktes über den Index in allen drei Arrays

Nachteil:

- Bestandteile eines Punktes nicht mehr erkennbar
- Fehleranfälligkeit der Indexverwaltung

Zusammenfassen der Elemente/Komponenten in einer **Struktur**.

„Sinn“ bleibt erhalten.

# Strukturen - Denkweise

---

- Welche Daten (ggf. unterschiedlichen Typs) bilden inhaltlich wieder eine Einheit?
- Aufbau von größeren, inhaltlich zusammenhängenden Strukturen



# Strukturen – Deklaration und Definition

---

Strukturen (Records) in C fassen mehrere Daten zusammen.

## Struktur-Deklaration:

```
struct <struct_name> {  
    <type> <element name>;  
    <type> <element name>;  
    <type> <element name>;  
};
```

## Beispiel:

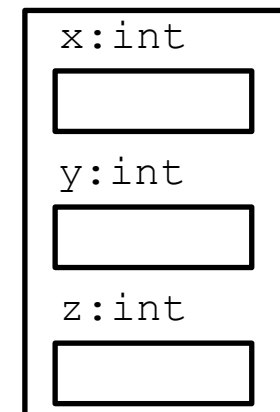
```
struct point {  
    int x;  
    int y;  
    int z;  
};
```

## Strukturvariablen-Definition:

```
struct <struct_name> <variablen_name> [ = {<initial>} ];  
start:struct point
```

## Beispiel:

```
struct point start;
```



# Strukturen - Initialisierung

---

Strukturen können initialisiert werden ähnlich zu Arrays über eine Liste mit Werten.

Beispiel Initialisierung:

```
struct point start = {1, 2, 3};
```

Die Initialisierung kann auch über die Angabe des Elements mit „.“ erfolgen.

Beispiel Initialisierung:

```
struct point start = {.z=3, .x=1, .y=2};
```

# Zugriff auf Komponenten

---

Auf die Elemente/Komponenten wird über den Punkt-Operator „.“ zugegriffen.

```
<variablen_name>.<element name> = <wert>;
```

**Beispiel:**

```
struct point start = {1, 1, 1};  
start.x = 5;  
u = start.y;
```



# Zugriff auf Komponenten

---

Auf Pointer innerhalb von Strukturen wird einfach über den Elementnamen zugegriffen.

Beispiel:

```
struct lookupTable { char ID[4]; int value; };
```

```
// Pointer in struct
```

```
struct lookupTable hash = {"HAW", 0};
```

```
hash.value = 1;
```

```
strncpy( hash.ID, "UAS", 3);
```

# Zugriff auf Komponenten

---

Wird eine Struktur über einen Pointer referenziert, so muss der „->“ Operator statt \* und . verwendet werden.

```
<poiner_name> -> <element name> = <wert>;
```

**Beispiel:**

```
struct point start = {1, 1, 1};
```

```
start.x = 5;
```

```
u = start.y;
```

```
// Pointer to struct
```

```
struct point *pointReference = &start;
```

```
pointReference->x = 10;
```

# Verwendung in Arrays

---

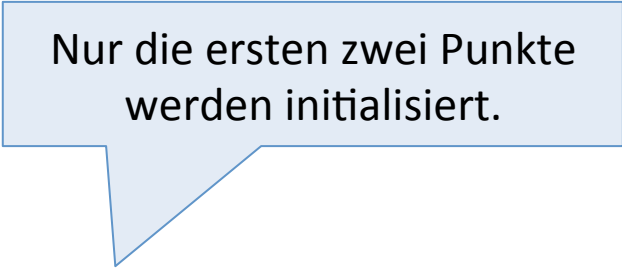
Strukturen bilden einen Datentyp und können auch in Arrays verwendet werden.

Beispiel:

```
struct point {  
    int x;  
    int y;  
    int z;  
};
```

```
struct point points[5] = {{0,0,0},{1, 1, 1}};
```

```
points[0].x = 5;
```



Nur die ersten zwei Punkte  
werden initialisiert.

# Typedef

Oftmals möchte man Strukturen als „echten“ Datentypen haben (ohne struct). Eigene Datentypen kann man mittels dem Schlüsselwort „typedef“ deklarieren/definiert.

```
typedef <type-information> <type name>;
```

## Variablen-Definition:

```
<type name> <variable name> [ = {<initial>} ];
```

Anonyme Struktur

## Beispiel:

```
typedef struct {  
    int x;  
    int y;  
    int z;  
} Points_t;
```

## Beispiel:

```
Points_t points[5] =  
{ {0, 0, 0}, {1, 1, 1}};  
  
points[0].x = 5;
```

Datentyp Bezeichner, beginnt mit Großbuchstaben.  
Oftmals Prefix „Type\_“ oder Suffix „\_t“ im Namen.

# Typedef

---

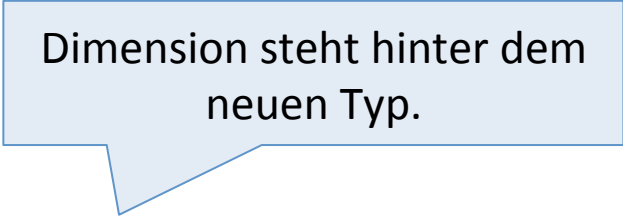
Typdefinitionen können auch für die Umbenennung von Standard-Datentypen verwendet werden, wenn sich dadurch die Lesbarkeit erhöht.

Beispiele:

```
typedef unsigned char Boolean;  
typedef unsigned char Matrikelnummer[7];
```

...

```
Matrikelnummer mat = {1,2,3,4,5,6};
```



Dimension steht hinter dem neuen Typ.