

Strukturen

- **Strukturtyp:** - selbst definierter Datentyp
- zusammengesetzt aus Komponenten *verschiedener Typen*
- Variable von Typ Struktur (Verbund/Record) kann Datensatz speichern

Personalnr.	Nachname	Vorname	Straße	Hausnr.	PLZ	Wohnort	Gehalt
-------------	----------	---------	--------	---------	-----	---------	--------

int	char[20]	char[20]	char[20]	int	int	char[20]	float
-----	----------	----------	----------	-----	-----	----------	-------

- *Komponenten* haben einen eigenen Namen und Typ (statt Index)

Deklaration eines Strukturtyps

- `struct name {
 komponententyp_1 komponentenname_1;
 komponententyp_2 komponentenname_2;
 :
 komponententyp_n komponentenname_n;
};`
- `struct` ist Schlüsselwort \rightsquigarrow **Datentyp**: `struct name`
- `name` ist Bezeichner (sog. Etikett)
- Anzahl der Komponenten bei Deklaration festgelegt
- Semikolon nach `}` (kein Anweisungsblock!)

Strukturvariablen

- **Definition:** `struct name variablenname;`
~> zusammengesetzte Variable vom Typ `struct name`
- besteht aus mehreren Komponentenvariablen
~> bei Strukturtypdeklaration vereinbart
- gleichzeitige Vereinbarung von Strukturtyp und -variablen möglich:

```
struct point {  
    float x;  
    float y;  
} punkt1;  
  
struct point punkt2, punkt3;
```

Beispiel

```
struct adresse {  
    char strasse[20];  
    int hausnummer;  
    int plz;  
    char wohnort[20];  
};
```

```
struct einwohner {  
    char name[20];  
    char vorname[20];  
    struct adresse anschrift;  
};
```

```
struct einwohner meyer, mueller;  
struct einwohner wohngebiet[100]; // Array aus 100 Einwohnern
```

Zugriff auf Komponentenvariablen

1. **Punktoperator:** *Strukturvariable.Komponentenvariable*

- Beispiele: `punkt1.x`
`meyer.name`
`meyer.anschrift.plz`

- lesender und schreibender Zugriff:

```
meyer.adresse.plz = 14482;  
strcpy(meyer.anschrift.wohnort, "Potsdam");  
printf("%5d %s\n",  
    meyer.anschrift.plz,  
    meyer.anschrift.wohnort);
```

2. **Pfeiloperator:** *Pointer_auf_Strukturvariable* → *Komponentenvariable*

- Beispiel: `(&punkt1) -> x`
`(&meyer) -> anschrift.plz`
- Pfeil: Minuszeichen und Größerzeichen
- lesender und schreibender Zugriff

Initialisierungslisten

- nur direkt bei der Definition der Strukturvariablen
- wie bei Arrays (mit Ausdrücken passenden Typs)
- Beispiel:

```
struct einwohner meyer = {  
    "Meyer",  
    "Peter",  
    { "Weberplatz",  
      98,  
      14482,  
      "Potsdam"  
    }  
};
```

Strukturen als Parameter und Rückgabewerte von Funktionen

- Voraussetzung: Deklaration des Strukturtyps *außerhalb* und vor den Funktionen
- Übergabe wie einfache Datentypen:
 ↪ komplett als zusammengesetzte Variable
- Alternative: Übergabe eines Pointers auf Strukturtyp
(call-by-value beachten!)

Ausblick: Dynamische Speicherverwaltung

- **Dynamische Variablen**

- kein Name, keine explizite Vereinbarung
- bei Bedarf zur Laufzeit durch Bibliotheksfunktionen angelegt
- Gültigkeit bis Speicherfreigabe, unabhängig von Programmstruktur

- **Biblioptheksfunktionen** zur dynamischen Speicherverwaltung

- in `<stdlib.h>`
- zum Anfordern von Speicher: liefern Pointer zurück, z.B.:
`void * malloc (size_t size)`
- zum Freigeben von Speicher: werden mit Pointer aufgerufen:
`void free (void * pointer)`

Komplexere Typen — Zusammenfassung

Beispiel	alpha ist ...
<code>int * alpha</code>	Pointer auf int
<code>float alpha[10]</code>	Array von 10 float-Komponenten
<code>char * alpha[20]</code>	Array von 20 Pointern auf char ↔ Array von 20 Strings
<code>char (* alpha) [10]</code>	Pointer auf ein Array von 10 char-Komponenten
<code>struct point alpha</code>	zusammengesetzte Variable vom Strukturtyp point (point muss zuvor definiert sein)

Vereinbarung eigener Typnamen

- Vereinbarung eines *Aliasnamens* für bereits deklarierte Datentypen
- `typedef Datentyp Aliasname;`
- `typedef int integer;`
- Anwendung:
 - Vereinfachung von Typnamen
 - * `typedef struct einwohner EINWOHNER;`
`EINWOHNER meyer;`
 - * `typedef unsigned long long ULL`
 - Vorbereitung Portierung maschinenabhängiger Datentypen
 - * `typedef int INT; ~\to typedef short INT;`

