Strukturen in C

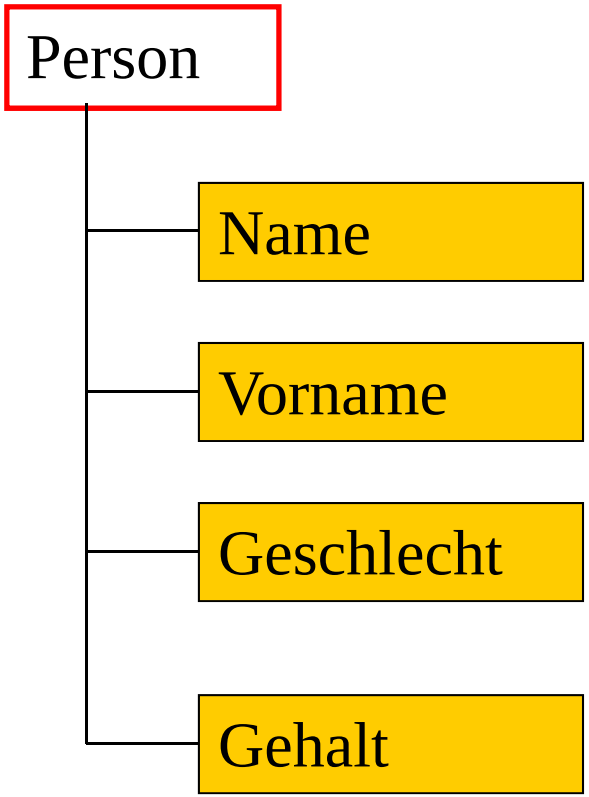
Strukturen stellen eine Zusammenfassung von Datenelementen unterschiedlichen Typs unter einem Namen dar.

Dadurch sind sie vergleichbar mit Feldern, die eine Zusammenfassung (Reihung) von Datenelementen gleichen Typs unter einem Namen darstellen.

Der Zugriff auf Datenelemente ist allerdings unterschiedlich:

- bei Feldern über den Index hinter dem Feldnamen, z.B. feld[i]
- bei Strukturen durch den Datenelementnamen hinter dem Strukturnamen und einem Punkt (“.“), z.B. Person.Name

Aufbau und Deklaration einer Struktur




```
struct  Person
 {
    char Name[30];
    char Vorname[30];
    int Geschlecht;
    float Gehalt;
};
```

Person ist der Strukturname (und damit wie ein Typ) und Name, Vorname,... sind Komponentennamen.

Aufbau und Deklaration einer Struktur (1)

Deklaration von Struktur-Variablen

Da die Deklaration einer Struktur wie eine Typdefinition wirkt, können Variable diesen Strukturtyps folglich unter Nennung von ***struct*** Strukturname deklariert werden.

Beispiel:

```
struct Person {...};
struct Person  p1,p2;
```

Es wäre auch eine Kombination möglich:

```
struct Person {...} p1,p2;
```

Aufbau und Deklaration einer Struktur (2)

Deklaration von Struktur-Variablen mittels Typ-Definition

typedef struct Person {...} person_t;

…

person_t p1,p2;

Ein per typdef geschaffener Typ kann wie ein Standarddatentyp (int, float, usw.) zur Deklaration von Variablen und zur Deklaration weiterer Strukturen benutzt werden.

Wertebelegung von Strukturvariablen

Wertebelegung von Struktur-Variablen:

Eine Möglichkeit besteht in der Initialisierung bei der Deklaration:

Beispiel: *struct Person*

p1={"Krause","Jens",1,3500.40},

p2={"Meier","Ines",0,4100.33};

oder es erfolgt eine Wertezuweisung an die Komponenten:

strcpy(p1.Name , "Krause");

p1.Gehalt = 3500.40;

Schachtelung von Strukturen

Strukturen können wieder Strukturen enthalten

Beispiel:

```
struct Datum
       { int tag;
         char monat[10];
         int jahr;
       };


 struct Person
      { char name[30];
        char vorname[20];
        struct Datum beginn;   /* Struktur innerhalb Struktur */
        int geschlecht;
      };
```

Schachtelung von Strukturen

Die Schachtelung darf aber nur in der angegebenen Weise erfolgen, d.h. man darf eine Struktur-Deklaration nicht direkt in eine weitere Struktur aufnehmen, sondern ausschließlich über eine Strukturvariable.

Zugriff auf geschachtelte Strukturen:
z.B.: struct person p1; und person enthält die geschachtelte Struktur struct datum beginn;

p1.name aber *p1.beginn.monat*

Felder von Strukturen

Strukturen können als Elemente in Feldern gereiht werden

Beispiel:

```
struct Person
   { char name[30];
     char vorname[20];
     struct Datum beginn;
      int geschlecht;
  };

struct Person mitarbeiter [5] ;
```

Zugriff: *mitarbeiter[1].name*
mitarbeiter[1].beginn.monat

Ein Beispiel mit Strukturen (1)

Nehmen Sie an, dass wir Werte an verschiedenen durch ihre Postleitzahl identifizierten Orten speichern wollen. Man könnte unabhängige Felder benutzen, um sowohl den Ort als auch die Postleitzahl zu speichern.

```
float werte[MAX_WERTE];
int plz[MAX_WERTE];
```

Hier bietet sich an, Wert und Ort in einer Struktur zusammenzufassen.

```
struct messung {
  float wert;
  char ort[MAX_STRLEN]; // hier jetzt der Ort statt PLZ, da wir schon
};                                  // mit Zeichenketten arbeiten können
…
struct messung m[MAX_WERTE]; // Feld
struct messung zws_messung; // Zwischenspeicher für Tausch
```

Ein Beispiel mit Strukturen (2)

```
i=0; ende=0; n_werte=0;
while(!ende)
{ if (i<MAX_WERTE)
  { printf("Eingabe Messwert Ort [%d]:",i);
    gets(eingabestring);
    sscanf(eingabestring,"%f", &m[i].wert);
    if (m[i].wert==999.0)
      ende=1;
    else
    { sscanf(eingabestring,"%s",dummystring);
      strcpy( m[i].ort, &eingabestring[strlen(dummystring)]);
      i = i + 1;
      n_werte ++;
    }
  }
  else // Gesamtanzahl der Messwerte erreicht
     ende = 1;
}
```

Ein Beispiel mit Strukturen (3)

Zuweisung von Strukturen, hier im Kontext des Tauschs der Elemente

….

```
if (tausch)
{  // Umspeichern der Strukturen
   zws_messung = m[i];
   m[i]  = m[i+1];
   m[i+1] = zws_messung;
}
```

Eine Zuweisung von Strukturvariable beinhaltet das Kopieren aller Komponenten auf die Zielstruktur. Hier werden auch die Zeichenketten kopiert.

Zuweisen, Kopieren von Strukturvariablen

Bei der Zuweisung von Strukturen a = b werden immer alle Elemente von b nach a kopiert.

Das ist gleichbedeutend mit Kopieren der Speicherinhalte, die eine Strukturvariable einnimmt.

Die Länge einer Struktur (hier kann der Typ oder die Strukturvariable benutzt werden) wird mit dem Operator

size_t sizeof (strukturname);

ermittelt. Vor dem C99-Standard wurde die Größe noch zur Übersetzungszeit ins Programm eingesetzt. Spätere Compiler erlauben, die Größe dynamisch zur Laufzeit zu bestimmen.

sizeof - Operator

size_t sizeof (name);

wobei *name* eine Variable, eine Struktur, ein Typ oder eine Feld sein kann. Namen für Felder werden hier aber nicht wie die Anfangsadresse auf das erste Element gewertet, sondern stehen für den Gesamtspeicherplatz des Feldes.

Beispiele:

```
#define MAX_STRLEN 120

struct messung {
      float wert;
      char ort[MAX_STRLEN];

};

struct messung mvar, m[10];

int l1 = sizeof(messung);   // ergibt 124
int l2 = sizeof(mvar);      // ergibt auch 124
int l3 = sizeof(m);         // ergibt 1240
```

Zeiger auf Strukturen (1)

Es können Zeiger auf Strukturen erklärt werden

Beispiel:

```
struct datum
   { int tag; char monat[10]; int jahr;
   } d1,d2;

struct datum *zeiger;

zeiger = &d1; // Adresse der Strukturvariablen d1

(*zeiger).tag  //Zugriff auf Komponente

zeiger->tag  //Zugriff auf Komponente
```

beide Schreibweisen sind identisch!

Zeiger auf Strukturen (2)

Das Kopieren von Strukturen, ausgehend von Zeigern:

Beispiel:

```
struct datum
{ int tag; char monat[10]; int jahr;
} d1,d2;

struct datum *zeiger1 = &d1; // Adresse der Strukt-Variablen d1
struct datum *zeiger2 = &d2; // Adresse d2

…

(*zeiger2) = (*zeiger1); // ist eine Möglichkeit

memcpy(zeiger2, zeiger1,sizeof(datum));
                              // die andere Möglichkeit
```

Strukturen an Funktionen übergeben

Es können Felder aus Strukturen als Parameter von Funktionenbenutzt werden (Feld, Call-by-Reference).

Beispiel:

```
int finde(int j, struct datum *p)
 { int i=0;
    while ((p->tag)!=0)
    {  if ((p->jahr)==j)  return (p->tag);
       else p++;
    };
    return 0;
 }
```

Aufruf:

```
struct datum d[5]={   {7,“April“,1981}, {12,“Juni“,1979},
                      {20,“Mai“,1980}, {4,“Juni“,1974},
                      {0,“ “,0}};

 t=finde(1980, d);  if (t) printf(“1980 am Tag %d\n“,t);
```

Strukturen an Funktionen übergeben

Einzelne Strukturvariablen können an Funktionen als Call-by-Value-Parameter vermittelt werden.

Beispiel:

```
int finde_datum( struct datum fd, struct datum *p)
 { int i=0;
   while ((p->tag)!=0)
    {  if ((p->jahr)==fd.jahr && p->tag==fd.tag &&
             strcmp(p->monat,fd.monat)==0)
            return (p->tag);
        else p++;
    }
    return 0;
}
```

Aufruf:

```
struct datum suchdatum={4,"Juni",1974};

t = finde_datum( suchdatum, d );
if (t) …
```

Strukturen als Rückgabe von Funktion

Einzelne Strukturvariablen können von Funktionen zurückgegeben

Beispiel:

```
struct datum ermittle_datum(int j, struct datum *p)
 { int i=0;
   struct datum leer={0,“ “,0};
   while ((p->tag)!=0)
    {  if ((p->jahr)==j)
            return (*p);
        else p++;
    }
    return leer;
}
```

Aufruf:

```
struct datum  ed =  ermittle_datum(1980, d );
if (ed.tag!=0) …
```