Übersicht

Lehrinhalt: Programmierung in C

- Überblick über Programmiersprachen, Allgemeines
- C-Basisdatentypen, Zahlendarstellung, Variablen, Konstanten
- Operatoren und Ausdrücke
- Anweisungen
- Kontrollstrukturen
- Funktionen
- Zeiger und Felder
- Zeichenketten (Strings)
- **Benutzerdefinierte Datentypen**
- Dynamischer Speicher
- Dateiarbeit
- Funktionspointer, Rekursion
- Preprozessor

Benutzerdefinierte Datentypen

Möglichkeiten des Programmierers:

- Umbenennung existierender Basisdatentypen per typedef

```
typedef int mytype;
mytype x, y;
```

- Aufzählungen

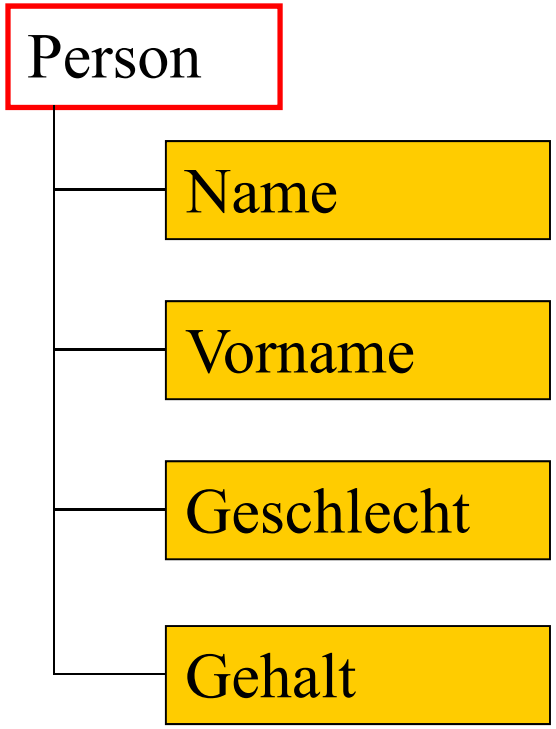
```
typedef enum {werktag, feiertag} tag_t;
tag_t heute;
```

- Strukturen:

  Zusammenstellung mehrerer unterschiedlicher Variablen zu einem übergeordneten Strukturtyp

Strukturen in C

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




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

Im Beispiel ist Person der Strukturname (und damit ein Typ), 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 ist 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 als Komponententyp weiterer Strukturen benutzt werden.

Zugriff auf Komponentenwerte

über den Namen der Strukturvariable, einen Punkt und den Namen der Komponente.

Beispiel:

```
struct person p1;
p1.gehalt = 2500;
p1.gehalt = p1.gehalt * 1.05;
```

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;
          int monat;
          int jahr;
        };


 struct Person
       { char name[30];
         char vorname[20];
         struct Datum gebdat; /* struct innerh. struct */
         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 gebdat;

`p1.name` und `p1.gebdat.monat`

Felder von Strukturen

Strukturen können als Elemente in Feldern gereiht werden

Beispiel:

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

struct Person mitarbeiter [5] ;
```

Zugriff:

```
mitarbeiter[0].name
mitarbeiter[0].gebdat.monat
```

Ein Beispiel mit Strukturen (1)

Nehmen Sie an, dass wir Werte an verschiedenen Orten speichern wollen. Hier bietet sich an, Wert und Ort in einer Struktur zusammenzufassen.

Deklaration des Struktur-Typs:

```
struct messung {
  double wert;
  char ort[MAX_STRLEN];
};
```

Deklaration von Variablen dieses Typs:

```
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,"%lf", &m[i].wert);
    if (m[i].wert==999.0)
      ende=1;
    else
    { sscanf(eingabestring,"%s",tempstring);
      strcpy( m[i].ort, &eingabestring[strlen(tempstring)]);
      i = i + 1;
      n_werte = n_werte+1;
    }
  }
  else // Gesamtanzahl der Messwerte erreicht
     ende = 1;
}
```

Vollständiges Programm: *strukturen.c*

Ein Beispiel mit Strukturen (3)

Zuweisung von Strukturen, hier im Kontext des Tauschs der Elemente

```
…
if ( m[i].wert>m[i+1].wert )
{
        // 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 {
    double wert;
    char ort[MAX_STRLEN];

};

struct messung mvar, m[10];

   int l1 = sizeof(messung);   // ergibt 128
   int l2 = sizeof(mvar);      // ergibt auch 128
   int l3 = sizeof(m);         // ergibt 1280
```

Zeiger auf Strukturen (1)

Es können Zeiger auf Strukturen erklärt werden

Beispiel:

```
struct datum
{ int tag, monat, jahr;
};

struct datum d1, d2;     // Zwei Variablen

struct datum *zeiger;    // Zeiger auf Struktur

zeiger = &d1; // Zuweisung der Adresse der
              //Strukturvariablen d1

(*zeiger).tag  //Zugriff auf Komponente

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

letzte beide Schreibweisen sind identisch!

Zeiger auf Strukturen (2)

Das Kopieren von Strukturen, ausgehend von Zeigern:

Beispiel:

```
 struct datum *zeiger1 = &d1;
// Adresse der Strukturvariablen 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 Funktionen benutzt werden (Call-By-Reference, d.h. als Zeiger).

Beispiel:

```
struct tupel { char zeichen; int nr;  };
struct tupel tups[4] = { { 'R', 1 }, { 'I', 3 }, { 'V', 60 }, {'H', 7}
};

char finde_tupel_zeichen(struct tupel *tp, int n, int such_nr)
{ int i=0;
  for (i = 0; i < n; i=i+1)
     if (tp[i].nr == such_nr)
        return tp[i].zeichen;
  return '\0';   // signalisiert, dass nichts gefunden wurde
}
```

Feld mit Strukturelementen

Aufruf:

```
suchnr = 7;
zeichen = finde_tupel_zeichen(tups, anz, suchnr);
if (zeichen != '\0')
  printf("Tupel mit nr %d und zeichen %c.\n", suchnr, zeichen);
```

Strukturen an Funktionen übergeben

Einzelne Strukturvariablen können an Funktionen als Call-by-Value-Parameter (hier *such_tupel*) vermittelt werden.

Beispiel:

```
int finde_tupel_index( struct tupel *tp, int n,
                       struct tupel such_tupel)
{
  int i=0;
  for (i = 0; i < n; i = i + 1)
     if ( tp[i].zeichen == such_tupel.zeichen &&
                       tp[i].nr == such_tupel.nr )
        return i;
  return -1;
}
```

Einzelnes Strukturelement

Aufruf:

```
index = finde_tupel_index(tups, anz, erzeuge_tupel('I', 3));
if (index > 0)
   printf("Gefunden an index %d: \n", index);
```

Strukturen als Rückgabe von Funktionen

Funktionen können einzelne Strukturvariablen mit ***return*** zurückgeben

Beispiel:

Strukturelement als Rückgabewert

```
struct tupel erzeuge_tupel(char zeichen, int nr)
{
   struct tupel neu;
   neu.zeichen = zeichen;
   neu.nr = nr;

   return neu;
}
```

Aufruf:

```
struct tupel tup_i3;
tup_i3 = erzeuge_tupel('I', 3);
```

Vollständiges Programm:
strukturen_u_funktionen.c