Felder in C

Felder stellen eine Reihung von Elementen gleichen Typs dar.
Man spricht auch von Vektoren oder Arrays.

Durch die Reihung (hintereinander speichern) kann ein Element über seine Nummer (Index) angesprochen werden

In C hat das erste Element den Index 0.

In C werden Felder durch die Verwendung der Indexklammern [ ] gekennzeichnet:

Beispiel: *int vektor[10];*

Deklaration von Feldern in C

Felder als eine spezielle Datenstruktur müssen deklariert werden. Die Deklarationsanweisung legt den Namen des Feldes, die Anzahl der Elemente und den Typ der Elemente fest.

Beispiel:

float a[10];

a ist der Name des Feldes
10 ist die Anzahl der Elemente (a_0 a_1 a_2 ... a_9)
float ist der Typ der Elemente

Deklaration von Feldern in C

float a[10];

Man beachte, dass die Anzahl der Elemente (Feldgröße) eine Konstante sein muss.

Bei Anwendungenmit von Fall zu Fall unterschiedlichen Feldgrößen muss das Feld trotzdem mit einer konstanten, maximal notwendigen Größe deklariert werden.

Innerhalb dieser maximalen Größe kann jeweils mit unterschiedlich vielen Elementen gearbeitet werden. Der Speicherplatz ist für die maximale Größe reserviert, aber es werden aktuell ggf. weniger Elemente belegt.

Initialisierung von Feldelementen

Feldelemente können bei der Deklaration der Feld-Variablen mit Werten belegt werden.

Beispiele:

```
double a[10]={-3.22,0.0,1,-7.234,55.5,6.6,-0.77,8,0.09,3};
```

```
int gewinnzahlen[6] = {1, 13, 17, 22, 34, 42};
```

In C ist es nicht möglich, die Größe des Feldes, bzw. die Anzahl belegter Elemente direkt anzusprechen (modernere Sprachen erlauben das).

Initialisierung von Feldelementen / Zugriff auf Feldelemente

Der Zugriff auf die Elemente eines Feldes erfolgt über den Index (Positionsnummer) des Elementes. Dieser Index wird in der Indexklammer angegeben.

Beispiel: es soll auf das 7.Element von a zugegriffen werden:

```
x = a[6];
```

(da das 1.Element den Index 0 hat, ist 6 der Index des 7.Elementes)

Bei Zugriffen auf Feldelemente ist stets zu prüfen, ob der Index in den Deklarationsgrenzen des Feldes liegt.

Beispiel eines Feldprogramms

```
#include <stdio.h>
#include <math.h>
//arithmetisches Mittel einer Messreihe
void main()
{ int i,n;
  float a[50],s;
  printf("\nEingabe Anzahl=");scanf("%d",&n);    // n<=50
  printf("\nEingabe des Vektors:\n");
  for (i=0;i<n;i++)                              //Eingabe der n Feldelemente
  {  printf("a[%2d]=",i);  scanf("%f", &a[i] );  }
  s=0;
  for (i=0;i<n;i++)
     s=s+a[i];                                   //Zählschleifen sind die typischen
  s=s/n;                                         //Steuerfluss-Konstr. für Felder
  printf("\n arithmetisches Mittel=%f \n",s);
}
```

Mehrdimensionale Felder

Mehrdimensionale Felder (z.B. Matrizen als 2-dimensionale Felder in der Mathematik) werden analog zum eindimensionalen Fall deklariert.

Eine zusätzliche Dimension wird durch eine weitere Indexklammer angegeben.

Beispiel: *float matrix[3][4];*

dies ist eine Matrix mit 3 Zeilen und 4 Spalten. Der Zeilenindex wird immer zuerst variiert. Diese Matrix hat die Struktur:

a_{00} a_{01} a_{02} a_{03}

a_{10} a_{11} a_{12} a_{13}

a_{20} a_{21} a_{22} a_{23}

Initialisierung mehrdimensionaler Felder / Zugriff auf Elemente

Mehrdimensionale Felder können auch in der Deklarations-anweisung initialisiert werden. Dabei muss die Verarbeitungs-reihenfolge Zeile -> Spalte usw. beachtet werden.

Beispiel: *float matrix[3][4]={ {1,2,3,4}, {5,6,7,8}, {9,10,11,12} };*

dies ist eine Matrix mit 3 Zeilen und 4 Spalten.

Der Zugriff auf Elemente erfolgt analog durch Angabe der Folge der Indexklammern in der Reihenfolge Zeile, Spalte, usw..

Da die Indizierung mit Null beginnt, selektiert die Angabe *matrix[2][3]* die dritte Zeile, und darin das vierte Element, also 12.

Mehrdimensionale Felder - Beispiel

Beispiel Matrix-Transposition:

```
#define N 10
float matrix[N][N];
float t;
int zeile, spalte;
// Eingabe Matrix
// …
for(zeile=0; zeile<N; zeile++)
{  for(spalte=zeile+1;spalte<N; spalte++)
   {   // Tausch matrix[zeile][spalte] mit matrix[zeile_neu][spalte_neu]
       // wobei zeile_neu = spalte und spalte_neu = zeile
       t = matrix[zeile][spalte];
      matrix[zeile][spalte] = matrix[spalte][zeile];
       matrix[spalte][zeile] = t;
   }
}
```