[Felder] Übersicht

- Felder, Rückblick
- Mehrdimensionale Felder

[Felder] Rückblick

- Vereinbarung von Feldern:

```
typ  name [anzahl];
typ name = {e1, e2, e3, ... , en}
```

- Die Adressierung von Feldelementen beginnt bei 0
- Die korrekte Adressierung muss der Programmierer selbst beachten
- Ein char-Vektor ist ein *String*, wenn eine Feldkomponente die ASCII-Null enthält
- Ein Feldname ist selbst keine Variable, sondern eine Zeigerkonstante (wird später erklärt)

[Felder] Feld von Feldern

Es können auch mehrdimensionale Felder gebildet werden (Feld von Feldern, Vektor von Vektoren)

z.B. zweidimensionales Feld (Matrix):

```
typ  name [anzahl-zeilen] [anzahl-spalten];
```

oder beliebig mehrdimensionale Felder um Tabellen abzubilden.

Wir können eine solche Matrix als Vektor betrachten, der seinerseits als Elemente Vektoren enthält.

[Felder] Adressierung von Feldelementen

- Der Programmierer muss selbst darauf achten, dass Feldangaben gültig sind.
  (keine Compiler-Warnungen)
- Feldadressierung:
  positive, ganzzahlige Indizes innerhalb der Feldgröße
  (0 ... n-1)
- Wie man Felder falsch auslesen kann (ohne dass ein Compiler warnt):

```
int m[2][3];

printf(“%d“,m[0][0];      /* erstes Element */
printf(“%d“,m[1][2];      /* letztes Element */
printf(“%d“,m[0][4]);     /* innerhalb des Array, aber richtiges Element ?*/
printf(“%d“,m[-1][3];     /* erstes Element, Absicht? ,*/
printf(“%d“,m[-1][0];     /* ausserhalb des Array */
printf(“%d“,m[2][0];      /* ausserhalb des Array */

m[0][3] = m[1][0];
```

[Felder] Initialisierung von Feldern

```
int v[6] = { 1 , 2 , 3 , 4 , 5 , 6 };

int v[] = { 1 , 2 , 3 , 4 , 5 , 6 };

int m[2][3] = { {1 , 2 , 3} , { 4 , 5 , 6} };

int m[][3] = { {1 , 2 , 3} , { 4 , 5 , 6} };

int m[4][2][3] = {{{ 1, 2, 3 },{ 4,5,6 }},
                  {{ 7, 8 }},
                  {{ 9 }, { 10 }}};

/ * letzteres entspricht
        {{{ 1, 2, 3} , {  4, 5, 6 )}},
         {{ 7, 8, 0} , {  0, 0, 0 )}},
         {{ 9, 0, 0} , { 10, 0, 0 )}},
         {{ 0, 0, 0} , {  0, 0, 0 )}}}  */

char str1[6] = "hallo";
char str2[]  = "hallo";
char str3[]  = {'h','a','l','l','o',\0'};
```

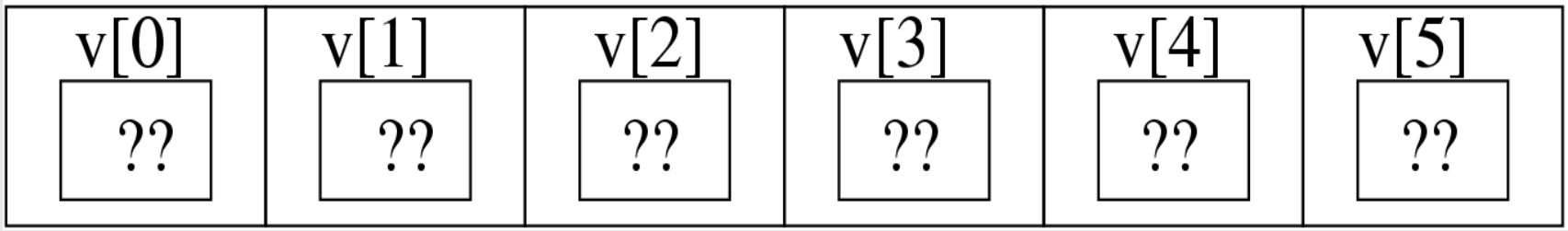
[Felder] Länge von char-Feldern

```
/* folgende Beispiele sind gleich gültig */

char str1[6] = "hallo";
char str2[]  = "hallo";
char str3[]  = {'h','a','l','l','o','\0'};

/*  Bei der Auswertung der Stringlänge durch
    strlen() wird das ASCII-Null nicht
    gewertet, sizeof hingegen bestimmt den
    den tatsächlichen Speicherbedarf */

strlen(str1) != sizeof(str1)

/* Nicht korrekt hingegen ist folgendes
    (wenngleich ds Programm ohne Warnung kompiliert */
    char str4[] = "welt";
    char str5[5]= "hallo";

    printf("%d %s %d\n",strlen(f1),f1,sizeof(f1));
    printf("%d %s %d\n",strlen(f2),f2,sizeof(f2));
```

[Felder] Anordnung von Feldern im Speicher

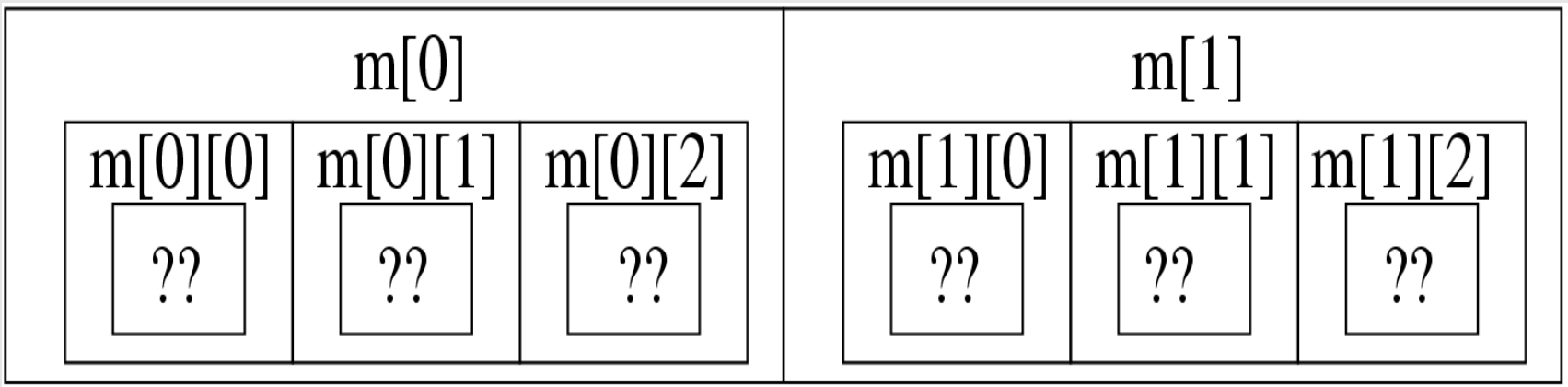
Vektor: 1-dimensionales Feld

```
int v[6];
```



Matrix: 2-dimensionales Feld

```
int m[2][3];
```



[Felder] Besonderheiten 2-dimensionaler Arrays

- Speicherintern ist ein n x m großes Array von float-Werten ein Speicherbereich von n * m Werten hintereinander gespeichert.

- Eine Matrix kann auch als Vektor von Vektoren aufgefasst werden.

```
float m[2][3]    : Matrix oder Vektor von Vektoren
m[0]             : bezeichent den ersten Zeilenvektor

float funktion1(float m[][3] , int zeilen);
                 : Die erste Dimension eines mehrdimensionalen Feldes
                   darf ohne Längenangabe sein.
                   Im obigen Beispiel könne aber nur Matrizen mit
                   Zeilen der Länge 3 übergeben werden.

float funktion2(float m[][], int spalten, int zeilen);
                 : ist nicht möglich, der Compiler kann ohne die
                   Zeilengröße die Speicherabbildungsfunktion nicht
                   berechnen.
```

[Felder] Adressierung von Feldelementen

```
#include <stdio.h>

#define YDIM 2
#define XDIM 3

int main()
{
	int m[YDIM][XDIM];
	int y,x;

	for ( y = 0; y < YDIM ; y++)
		for (x=0; x < XDIM ; x++)
			m[y][x]=y*XDIM+x;

	for ( y = 0; y < YDIM ; y++)
	{
		for (x=0; x < XDIM ; x++)
		{
			printf("%d ",m[y][x]);
		}
		printf("\n");
	}
	return 0;
}
```

```
0  1  2
3  4  5
```

[Felder] Falsche Adressierung

- Fehlermeldung beim Programmablauf wie
  - Segmentation fault
  - Bus error
  - memory fault

  sind nicht allein auf array-Probleme zurückzuführen, aber oft genug ein Hinweis auf solche.

[Felder] Felder als Parameter

- Ein String als Sonderform eines Feldes ist begrenzt durch den Stringanfang und das Nullbyte '\0'.
- Liest man einen String Zeichen für Zeichen, kommt man schließlich zu \0 und damit zum Ende
- Ein String von n Zeichen Länge belegt wegen des '\0' (n+1) * sizeof(char) Arbeitsspeicher
- Das gilt nicht für andere Felder
- Benötigt eine Funktion die Länge eines Feldes, mit welchem es operieren soll, so muß diese übergeben werden (Ausnahme: char-Felder)

[Felder] Felder als Parameter: Beispiel

```
float vektorsumme(const float v[],int laenge)
{
    float s = 0;
    while(--laenge >= 0)
        s+= v[laenge];
    return s;
}
```

```
float v1[7], v2[50], m[5][20], summe;

summe = vektorsumme(v1,7) + vektorsumme(v2,50);
summe = vektorsumme(m[3],20);

summe = vektorsumme(v1,10);   /* feldindex-Ueberschreitung ! */

summe = vektorsumme(v2,10);
summe = vektorsumme(&v2[10],10);
summe = vektorsumme(m[0],100);
```