9. Felder (Arrays)

Felder dienen zur gemeinsamen Speicherung gleicher Datentypen.

9.1. Eindimensionale Felder

Beispiel: Deklaration eines Integer-Feldes mit 5 Elementen

```
int n[5];
```

Diese 5 Variablen liegen dann im Speicher hintereinander:

n[0]	n[1]	n[2]	n[3]	n[4]

Wichtig: Die Angabe der Feldgröße muss in diesem Fall durch eine Konstante erfolgen, deshalb die Bezeichnung "statisch"! Die so deklarierten Felder beginnen immer mit Index `[0]` und enden mit Index `[`*Feldgröße*`-1]`.

Es erfolgt **keine Überprüfung auf gültigen Speicherbereich** von Seiten des Compilers!

Ansonsten gilt für Feldelemente das gleiche, wie für sonstige Variable dieses Typs. Bei der Deklaration kann eine elegante Initialisierung erfolgen.

Beispiel:

```
int n[] = {8,7,9,-13};
```
int [0] = 8;
int[1] = 7;
int[2] = 9;
int[3] = -13;

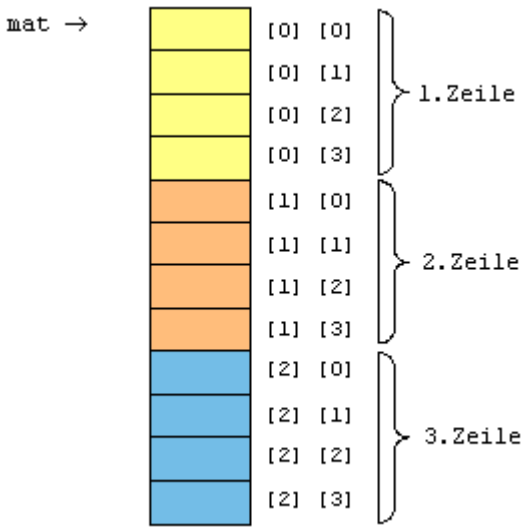
Bsp. Deklarieren Sie ein Feld der Größe 100 und initialisieren Sie diesen mit der Zahl 0.

9.2. Mehrdimensionale Felder

Bei mehrdimensionalen Arrays werden die Variablen durch ein Tupel von Indizes unterschieden. Beispiel (Deklaration):

```
double x[8][30];
```

Hier gilt dann analog: erstes Element ist `x[0][0]`, bzw. letztes `x[7][29]`. Dementsprechend werden zweidimensionale Array zeilenweise gespeichert:



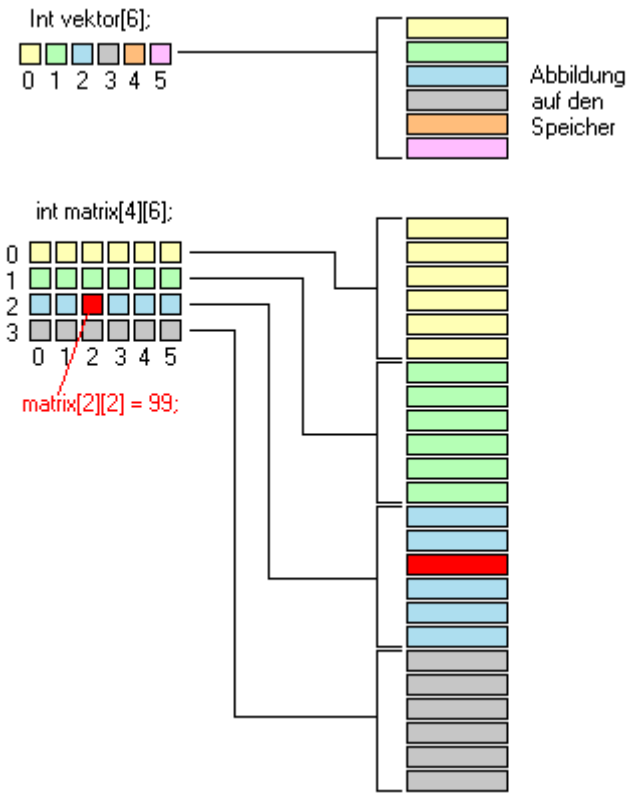
Analog lassen sich auch Felder mit mehr als zwei Dimensionen definieren. Angesprochen werden die Komponenten über den Index:
`x[8][30] =12.5;`

Anstelle der Zahlen kann für den Index auch ein beliebiger Integer-Ausdruck angegeben werden, z. B.:
`x[i-1][j*k+2] =32.5;`

Der Compiler kann aber selbstständig nur die Zahl der Zeilen feststellen; die notwendige (Mindest-)Zahl der Spalten muss ihm mitgeteilt werden.

Bemerkung: *intern* werden mehrdimensionale Felder (wenn statisch allociert!) eindimensional angelegt; wenn also z. B. ein Array deklariert ist mit `int a[4][3];`, dann sind die Aufrufe `a[i][j]`, `a[0][i*3+j]` und `a[k][(i-k)*3+j]` völlig äquivalent:

a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]	a[2][0]	...
a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0][5]	a[0][6]	...



Auch hier gibt es bei der Deklaration wieder die bequeme Möglichkeit der Initialisierung:

```
int md[5][10] =
  {
   {0,1,2,3,4,5,6,7,8,9},
   {1,2,3,4,5,6,7,8,9,10},
   {2,3,4,5,6,7,8,9,10,11},
   {3,4,5,6,7,8,9,10,11,12},
   {4,5,6,7,8,9,10,11,12,13}
  };
```

Die Angaben für die zweite (und weitere) Dimension werden also jeweils in eigene geschweifte Klammern geschrieben. Innerhalb der geschweiften Klammern gilt, dass eventuell nicht initialisierte Elemente auf null gesetzt werden. Die Kennzeichnung der Array-Struktur bei den Initialisierungswerten ist notwendig, wenn Elemente "innerhalb" des Arrays durch fehlende Angabe mit 0 initialisiert werden sollen. Beispiele:

```
int mat[3][4] = {  {  1,  4,  3, -4 },
                   {  2,  0, -3,  1 } };
```

Die letzte Zeile (`mat[2][0] ... mat[2][3]`) wird mit 0 initialisiert.

```
int mat[3][4] = { {  1,  4,  3 },
                  {  2,  0, -3 },
                  {  0, -5,  6 } };
```

Die letzte Spalte (`mat[0][3], mat[1][3], mat[2][3]`) wird mit 0 initialisiert. Die Kennzeichnung der Arraystruktur bei den Initialisierungswerten ist notwendig.

Man kann die inneren geschweiften Klammern auch weglassen, dann wird der ganze Vektor Element für Element initialisiert.

```
int mat[3][4] = { 1,  4,  3, -4,  2,  0, -3,  1,  0, -5 };
```
oder:
```
int mat[ ][4] = { 1,  4,  3, -4,  2,  0, -3,  1,  0, -5 };
```
Die beiden letzten Elemente der letzten Zeile (`mat[2][2], mat[2][3]`) werden mit 0 initialisiert.

Beispiel
Deklarieren Sie ein Feld der Größe 30 bzw 30 * 30 und initialisieren sie diese mit der Zahl 0.