

## 4. Vektoren (Arrays, Felder)

Oft ist es erforderlich, in einem Programm eine Menge von Variablen zu deklarieren, z.B. bei einer Datenbankverwaltung. Hierfür gibt es die Möglichkeit, einen sog. **Vektor**, das ist ein ganzes Feld von Variablen, zu deklarieren. Man nennt solche Variablen auch **indizierte Variablen**. Bei einer Vektor-Deklaration wird nach dem Feldnamen in eckigen Klammern die Größe, d.h. die Anzahl der Elemente angegeben, wie folgendes Beispiel zeigt :

```
int zahl [10];
```

Mit dieser Deklaration wird ein Feld, bestehend aus 10 Variablen vom Typ *int* deklariert :

zahl[0]	zahl[1]	zahl[2]	zahl[3]	zahl[4]	zahl[5]	zahl[6]	zahl[7]	zahl[8]	zahl[9]
---------	---------	---------	---------	---------	---------	---------	---------	---------	---------

## Strukturierte Datentypen

Im Programm spricht man die einzelnen Variablen mit dem Feldnamen und ihrem Index (=Elementnummer) an, z.B.

```
zahl [5] = 150;  
oder :   scanf("%d", &zahl [3]);
```

Wichtig: Im Unterschied zu anderen Programmiersprachen **beginnt in C jeder Vektor mit der Elementnummer 0 !**.

Für diese Elementnummer kann man im Programm eine Zählvariable einsetzen und so mit einer einzigen Anweisung innerhalb einer *for*-Schleife alle Variablen bearbeiten, z.B.

```
for (i=0; i<=9; i++) scanf ("%d",&zahl [i] )
```

Die Anzahl der Feldelemente eines Vektors wird meistens zu Programmbeginn durch eine Konstante festgelegt, z.B. :

```
# define MAXIMUM 100  
...  
double x [MAXIMUM];
```

## Strukturierte Datentypen

Ebenso wie einfache Variablen können auch Vektoren bei der Deklaration initialisiert werden :

```
int a [ ] = {10, 20, 30, 40, 50};
```

Hierbei wird den Elementen `a[0]`, `a[1]`, `a[2]`, `a[3]` und `a[4]` der Reihe nach die Zahlen 10, 20, 30, 40 und 50 zugewiesen. Bei gleichzeitiger Initialisierung muss bei der Deklaration keine Feldgröße angegeben werden. Der Compiler legt hier automatisch die Größe des Feldes fest (diese entspricht der Anzahl der Elemente!

Neben den eindimensionalen Vektoren sind auch mehrdimensionale Vektoren möglich :

```
int x [4] [3];
```

Diesen zweidimensionalen Vektor kann man sich vorstellen als Matrix bestehend aus 4 Zeilen und 3 Spalten :

## Strukturierte Datentypen

x [0] [0]	x [0] [1]	x [0] [2]
x [1] [0]	x [1] [1]	x [1] [2]
x [2] [0]	x [2] [1]	x [2] [2]
x [3] [0]	x [3] [1]	x [3] [2]

Eine besondere Rolle spielen eindimensionale char-Vektoren, um **Zeichenketten** zu speichern:

z.B. **`char nachname [20];`**

Die Initialisierung eines char-Vektors : `char nachname[] = {'H', 'U', 'B', 'E', 'R'};` wird abgekürzt, indem man anstelle der einzelnen Zeichen in Klammern eine Zeichenkette angibt. Eine Zeichenkette muss dabei immer in doppelte Hochkomma stehen :

**`char nachname[] = {"Huber"};`**

Nach dieser Initialisierung besteht der Vektor aus den folgenden **6** Elementen :  
`nachname[0] = 'H', nachname [1] = 'u', nachname[2] = 'b', nachname [3] = 'e',  
nachname[4] = 'r' nachname [5] = '\0'`

Der C-Compiler markiert stets das Ende eines *char*-Vektors mit dem sog. Nullzeichen `'\0'`, damit beim Programmablauf das Ende der Zeichenkette erkannt werden kann.

## Strukturierte Datentypen

**Funktionen zur Ein-/ Ausgabe eines Strings** (Beispiel: `char name [25];` ):

***gets* (name);** liest eine Zeichenkette von Tastatur ein und hängt ein sog. String-Endezeichen `\0` am Ende an. Z.B. belegt der String "Huber" 6 Elemente im Vektor name:  
`name[0] = 'H', name[1] = 'u', name[2] = 'b', name[3] = 'e', name[4] = 'r', name[5] = '\0',`

***puts* (name);** gibt die Zeichenkette in name bis zu einem `\0` am Bildschirm aus. Sie endet mit einem Zeilenvorschub

***printf* ( "%s", name);** das Formatsteuerzeichen `%s` wird für Stringausgabe verwendet. Hier ist im Vergleich zur *puts*-Funktion eine Formatierung möglich

***scanf* ("%s", name);** die *scanf*-Funktion kann mit dem Formatsteuerzeichen `%s` eine Zeichenkette einlesen. Problem: sie liest den Enter-Code aus dem Eingabe-Puffer im Vergleich zur *gets*-Funktion nicht mit aus.

Zu beachten: der Name eines Vektors liefert die Anfangsadresse des Vektors! Deshalb darf in der *scanf*-Funktion kein `&` angegeben werden!

## wichtige String-Funktionen:

***strcpy*** (*name1*, *name2*); kopiert den String *name2* nach *name1*. Die Wertzuweisung *name1=**name2* ist in C nicht möglich!

***strcat*** (*name1*, *name2*); verkettet die beiden Strings und schreibt den zusammengesetzten String in *name1*. *name1* muss dazu ausreichend groß sein!

## wichtige String-Funktionen:

***strlen*** (*name*); liefert die Länge der in *name* gespeicherten Zeichenkette ohne '\0', z.B.  $le = strlen(name)$ ; ergibt für *name*="Huber" die Länge  $le = 5$

$a =$ ***strcmp*** (*name1*, *name2*);

vergleicht 2 Strings (lexikalisch):

ist *name1* gleich *name2* dann ist  $a = 0$

ist *name1* < *name2* dann ist  $a < 0$

ist *name1* > *name2* dann ist  $a > 0$

Die Funktion *strcmpi* unterscheidet dabei nicht zwischen Groß- und Kleinschreibung.