6 Felder

Allgemein

Felder/Arrays dienen zum Speichern von mehreren Variablen gleichen Datentyps. Die Deklaration ist:

```
int arr[20];
```

Damit wird ein Feld erzeugt bestehend aus 20 Integer-Elementen. Die Größe eines Feldes kann auch in C nicht nachträglich verändert werden.

Achtung: bei der Defintion wird die eckige Klammer hinter dem Variablennamen angegeben. Nicht wie z.B. in Java **int[] arr;**. Das wäre logischer, man kann dann lesen: arr ist vom Typ **int[]** - Integer-Array.

Der Zugriff auf die Elemente (Lesen oder Schreiben) erfolgt mittels Index in der eckigen Klammer:

```
arr[3] = 7;
```

Der 1te Index ist 0 und der höchste Index ist für obiges Beispiel 19.

Die Sprache C hat kein ausgefeiltes Schutzkonzept wie moderne Programmiersprachen. Ein sehr häufiger Fehler ist es in Feldern über die Feldgrenzen hinaus in ein Feld zu schreiben. Es gibt keine Exception oder Fehlermeldung. Es wird einfach im nicht reservierten Speicher geschrieben/gelesen. Dabei können durchaus völlig andere Variable unbewusst überschrieben werden.

Daher ist darauf zu achten die Feld-Grenzen unter keinen Umständen zu verletzen.

```
int c[20];
c[47] = 0xFFFFF;        // deutlich größer als der größte Index 19 --> KEINE FEHLERMELDUNG es wird irgendwas ueberschrieben
```

Empfehlung:

```
#define ANZELEM 10
int i;
int arr[ANZELEM];

for (i = 0; i < ANZELEM; i++) {...}
```

Einfach ein Makro verwenden um die Feldgröße zu definieren und darauf zuzugreifen. Ist wesentlich besser zu Lesen als *magische Zahlen* die im Code verteilt sind und die Feldgröße kann so auch an einer einzigen Stelle verändert werden.

Auch in C gibt es eine Abkürzungen für die Initialisierung:

```
int c[] = {1,2,3};
```

Deklariert ein 3-großes Feld und weißt ihm die Werte 1, 2 und 3 zu.

In C werden Variable nicht automatisch bei der Deklaration initialisiert. Das bedeutet in einer neu deklarierten Variable steht was immer vorher im Speicher an der entsprechenden Stelle gestanden ist. Eine komfortable Art ein Feld auf 0 zu initialisieren ist:

```
int c[4] = {0};         // damit gilt c[0] = c[1] = c[2] = c[3] = 0
int a[4] = {1, 2, 3}    // damit gilt a[0] = 1   a[1] = 2    a[2] = 3    a[3] = 0
```

Nach der obigen Deklaration und Initialisierung werden die kompleten Felder initialisiert, nicht nur die erwähnten: c = [0, 0, 0, 0] und a = [1, 2, 3, 0]. Die nicht angeführten Elemente werden automatisch mit 0 initialisiert.

Größe des Feldes

C ist nicht objektorientiert. Es kann daher kein c.Length oder ähnliches geben. Um die Anzahl der Elemente in einem Feld zu bestimmen kann die sizeof() Funktion verwendet werden. Diese Funktion gibt immer die Größe der Variable in Bytes an. Für Felder bedeutet das Anzahl der Feldelemente mal Größe eines Elements:

```
int c[30];
int groesse = sizeof(c);                // Speicherbedarf des gesamten Feldes in Anzahl von Bytes
int numElem = groesse / sizeof(c[0]);   // Division durch Speicherbedarf eines einzelnen Elements
                                       // ergibt die Anzahl der Elemente
```

Auch daher beste Variante ist das Feld über Konstante oder Makros zu definieren:

```
#define NUMELEM 10                 // Makro NUMELEM=10   (~Konstante)
int c[NUMELEM];                   // 10 Elemente

for (int i = 0; i < NUMELEM; i++)   // Durchlauf bis 10
  c[i] = i;
```

Dadurch ist ein einfaches Verändern der Feldgröße im Nachhinein einfach und fehlervermeidend möglich.

CHAR-Felder

Zeichen-Felder sind in C besonders wichtig. Sie werden verwendet um Zeichenketten/Strings zu speichern (es gibt ja keinen String-Typen).

```
char myString[] = "Hallo";
```

myString ist ein char-Feld. Mit einer String-Konstante (Gänsefüßchen) kann das Feld abgekürzt initialisiert werden.

> Ganz spezielle Konvention: Zeichenketten werden immer mit einem EOS (End of String) Zeichen abgeschlossen ('\0' bzw. ASCII-Code 0). Viele Standardfunktionen setzen das voraus (printf gibt Zeichenkette aus bis '\0' gefunden wird).

Daher hat myString die Größe 6. 5 für die sichtbaren Zeichen und 1 zusätzliches für das EOS-Zeichen das automatisch eingefügt wird.

Ausgegeben kann der String so werden:

```
for (i=0; myString[i] != '\0'; i++)         // Ausgabe bis EOS
  printf("%c", myString[i]);                // oder: putchar(einText[i]);
```

einfacher geht's so:

```
printf("mein String: %s\n", myString);
printf(myString);
```

Damit wird auch deutlich für was das '\0' Zeichen notwendig ist - die Funktionen wie *printf* könnten ja sonst nicht die Zeichen bis zum Stringende ausgeben. In Objektorientierten Sprachen kann ein String-Objekt die Länge-Eigenschaft besitzen, die ausgewertet werden kann.

Wesentlich kritischer ist das Einlesen von Zeichenketten. Die Funktionen scanf(), getch() oder fgetc() funktionieren natürlich, allerdings sind diese Funktionen nicht für Zeichenketten entworfen worden und fügen daher nicht automatisch ein EOS ein. Wird auf diese Art in eine Zeichenkette eingelesen muss man selbstständig nach dem Einlesen ein '\0' ans Ende setzen.

Um Fehler zu vermeiden ist die String-Methode **fgets(ziel, laenge, stream)** vorzuziehen:

```
#define LAENGE 10
char myString[LAENGE];          // 10 lange Zeichenkette, kann 9 Zeichen + '\0' speichern
fgets(myString, LAENGE, stdin);  // egal wieviel eingegeben wird, nur 9 Zeichen werden in die Variable myString geschrieben
```

Bei einer Eingabe länger als 9 Zeichen werden die überzähligen Zeichen einfach weggeschnitten und an der 10ten Stelle automatisch ein EOS eingefügt. Der zweite Parameter der fgets-Funktion muss exakt der Länge der Zeichenkette entsprechen.

Die Eingabe wird mit der Eingabetaste abgeschlossen. Wird eine Zeichenkette kürzer als die Länge des Feldes eingegeben, dann steht auch der Zeilenumbruch im Feld. Zum Löschen kann dieses letzte Zeichen mit einem EOS überschrieben werden:

```
myString[strlen(myString)-1]='\0'
```

Die Länge eines Strings (ohne '\0') kann mit der **strlen()**-Funktion ermittelt werden. Dabei wird die Länge bis zum EOS zurückgegeben.

String-Funktionen

- **Kopieren einer Zeichenkette** (quellstr) in eine Andere (zielstr). Die Zielzeichenkette muss groß genug sein für die Quellzeichenkette (samt EOS):

  strcpy(zielstr, quellstr)

  Besser:

  strncpy(zielstr, quellstr, n)

  Damit werden maximal n Zeichen kopiert. n setzt man auf die Größe vom zielstr.

- **Vergleich zweier Zeichenketten**. Sind beide Strings gleich, gibt diese Funktion 0 zurück. Ist der String str1 kleiner (früher im Alphabet) als str2, ist der Rückgabewert kleiner als 0; und ist str1 größer (später im Alphabet) als str2, dann ist der Rückgabewert größer als 0 :

  int strcmp(str1, str2)

  Besser:

  strncmp(str1, str2, n)

  Damit werden maximal n Zeichen verglichen. n setzt wird auf die Größe von str1 gesetzt.

- **Typkonvertierungen** nach String. Genau wie für printf() kann mit sprintf() in eine Zeichenkette geschrieben werden:

  sprintf(str, "Die Zahl %i ist größer als die Zahl %i", 7, 3);

  Das Ziel dieser Schreiboperation ist halt in diesem Fall anstatt eines Streams (Bildschirm ...) die Zeichenkette str. str muss natürlich zuvor so groß definiert werden, dass die Zeichenkette reinpasst.

  nach Zahl. "**A**scii-**To**-**I**nteger"

  int **atoi**(const char *str)

  "**A**scii-**To**-**F**lieszkomma"

  double **atof**(const char *str)

- **Lesen aus einer Zeichenkette** funktioniert wie einlesen von der Tastatur mit scanf(). Hier wird die Zahl aus der Zeichenkette str in die int-Variable num gelesen:

  sscanf(str, "%i", &num);

Mehrdimensionale Felder

Mehrdimensionale Felder:

```
int a[3][2];
```

In C können natürlich völlig analog auch mehrdimensionale Felder definiert werden. In C gibt es nur rechteckige Felder (im Gegensatz zu Java oder C#).