4.3 Datenstrukturen/Arrays

4.3.1 Beispiel

Eine Folge von endlich vielen zusammengehörenden Daten gleichen Typs heißt Array[1]. Wie viele andere Programmiersprachen auch stellt C die Mittel bereit, um ein Array zu definieren und zu benutzen.

Das folgende Programm nimmt eine Folge von Messwerten von der Tastatur (oder aus einer Datei) auf und berechnet daraus einige Kennwerte:

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      double arithmittel ;
6      int    lauf        ;
7      double messwert[10];
8      double summe       ;
9
10     messwert[0]=47.21;
11     messwert[1]=46.97;
12     messwert[2]=47.80;
13     messwert[3]=48.12;
14     messwert[4]=47.15;
15     messwert[5]=48.19;
16     messwert[6]=45.02;
17     messwert[7]=44.72;
18     messwert[8]=49.35;
19     messwert[9]=45.28;
20
21     summe=0.0;
22     for(lauf=0; lauf<10; ++lauf)
23     {
24         summe = summe + messwert[lauf];
25     }
26
27     arithmittel = summe/10;
28
29     printf("Summe der Messwerte ......: %lf\n", summe);
30     printf("Arithmetischer Mittelwert: %lf\n", arithmittel);
31
32     return 0;
33 }
```

4.3.2 Definition

Hier wird ein Array mit dem Namen `messwert` definiert.

```
1      double messwert[10];
```

Jedes Element dieses Arrays hat den Typ `double`. In eckigen Klammern befindet sich die Größe des Arrays, nämlich die Anzahl der Elemente. Hier befindet sich eine positive, ganzzahlige Konstante (0 ist nicht erlaubt).

[1] oder Feld oder Vektor oder Reihung

Bis zur Version C89 ist eine Festlegung der Arraygröße durch den Inhalt einer Variablen *nicht* möglich. In C99 und C11 gibt es allerdings sogenannte VLAs (*variable length arrays*). Bei ihnen ist es erlaubt, als Anzahl eine Variable zu verwenden, falls die Variable in einem äußeren Block definiert wurde. Diese Erweiterung ist jedoch umstritten, weil C damit inkompatibel zu C++ wird. Deshalb wird hier nicht weiter darauf eingegangen.

4.3.3 Zugriff

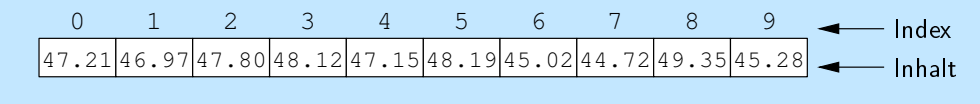

Abbildung 1: Array, Inhalt und Index

Der Zugriff auf ein Element des Arrays findet wie folgt statt:

```
1   summe = summe + messwert[lauf];
```

Mit dieser Schreibweise wird jeweils nur auf *ein* Element des Arrays zugegriffen, und zwar auf das Element mit der Nummer `lauf`. Der Ausdruck in eckigen Klammern wird auch *Index* genannt. Der Index darf (wie hier) variabel sein, so dass zur Laufzeit des Programmes bestimmt werden kann, auf welches Element zugegriffen werden soll. Bei C hat das erste Element immer den Index (die Nummer) null (siehe Abbildung 1).

4.3.4 Arraygrenzen

In C wird nicht überprüft, ob bei einem Zugriff die Arraygrenzen eingehalten werden. Ausdrücke wie `messwert[-4]` oder `messwert[12345]` sind erlaubt, führen aber beim Fehlschlag zu undefiniertem Verhalten des Programms (falsche Ergebnisse, unerklärliche Abstürze usw.).

4.3.5 Zuweisung ganzer Arrays

Eine Zuweisung eines Arrays an ein anderes ist in C (anders als etwa in PASCAL) nicht möglich:

```
1   int a[3];
2   int b[3];
3   a=b;
```

Stattdessen muss man jedes Element einzeln zuweisen:

```
1   int a[3];
2   int b[3];
3   int index;
4   for(index=0; index<3; ++index)
5       a[index]=b[index];
```

4.3.6 Vergleich ganzer Arrays

Ein Vergleich eines Arrays mit einem anderen Array funktioniert in C ebenfalls nicht:

```
1   int a[3];
2   int b[3];
3   if(a==b)
4   { ... }
```

Hier schlägt der Vergleich immer fehl, selbst wenn alle Elemente gleich sind[2]. Hier muss man also jedes Element einzeln vergleichen:

```
1  int a[3];
2  int b[3];
3  int a_gleich_b=1;
4  for(index=0; index<3; ++index)
5      if(a[index]!=b[index])
6          a_gleich_b=0;
7  if(a_gleich_b)
8  { ... }
```

4.3.7 Initialisierung von Arrays

Anstelle der zehn Zuweisungen in den ersten Zeilen des Beispielprogramms hätte man die Variable `messwert` schon bei der Definition initialisieren (=mit Startwerten versehen) können.

Eine Initialisierung ist in C nicht dasselbe wie eine Zuweisung. Die Initialisierung ist erlaubt, die Zuweisung nicht. Die Initialisierung erfolgt der Reihe nach durch eine Komma-getrennte Liste aus konstanten Werten. Sie ist durch geschweifte Klammern begrenzt:

```
1  double messwert[10]={47.21,46.97,47.80,48.12,47.15,
2                       48.19,45.02,44.72,49.35,45.28};
```

Hat die Liste weniger Elemente als das Array, dann wird damit nur der vordere Teil des Arrays mit den aufgelisteten Werten initalisiert. Der Rest ist dann nicht undefiniert, sondern wird mit 0-Werten belegt. Bei Ganzzahlen ist das `0` (bzw.`'\0'`), bei Gleitkommazahlen ist es `0.0` und bei Zeigern `NULL`[3].

```
1  double messwert[10]={47.21,46.97};
```

In diesem Beispiel werden die Elemente Nr. 0 und Nr. 1 mit den in der Liste angegebenen Werten gefüllt, während die Elemente Nr. 2 bis Nr. 9 den Wert `0.0` bekommen.

Bei C99 und C11 ist es übrigens möglich, die Liste gezielt per Index (*designator*) zu füllen. Im nächsten Beispiel ist es das Element Nr. 4, das gezielt angesprochen wird. Danach geht die Liste weiter mit dem folgenden Element (Nr. 5):

```
1  int prim[7]={29,31,[4]=37,41};
```

Hier hat das Array anschließend den Inhalt `{29,31,0,0,37,41,0}`.

Wenn man ein Array initialisiert, kann man die Anzahl der Elemente weglassen. Der Compiler berechnet dann die Anzahl so, dass sie alle aufgelisteten Werte umfasst:

```
1  int a[]={16,33,45,78};   /* 4 Werte                  */
2  int b[]={3,[40]=9};      /* Nur C99+C11: 41 Werte    */
3  int c[]={[8]=1,1,[0]=1};/* Nur C99+C11: 10 Werte    */
4  int d[]={12345};         /* 1 Wert                   */
5  int e[]={};              /* Nicht erlaubt            */
6  printf("a hat %u Bytes\n", sizeof(a));
```

In der letzten Zeile des obigen Beispiels wird der Operator `sizeof()` benutzt. Mit ihm kann man die Speichergröße einer Variablen in Bytes erhalten.

Wenn man ein Array nicht initialisiert hat, haben alle Elemente zufällige Werte, wie man es von Zeigern und einfachen Datentypen her auch kennt[4].

[2] Woran das liegt, kann man in einem späteren Kapitel erfahren.
[3] Sind die Elemente wiederum Arrays, Records oder Unions, werden deren Bestandteile auf 0 gesetzt.
[4] Ausnahmen sind wie immer globale und statische (dazu später) Variablen.