

## L9. Arrays und Strukturen

- Eindimensionale Arrays
- Strings
- Zeiger, Adressen und Arrays
- Zeigerarithmetik
- Übergabe von Zeigern an Funktionen
- Mehrdimensionale Arrays
- Strukturen und Strukturvariablen
- Aufzählungstypen

### Eindimensionale Arrays

**Arrays** (auch **Vektor**, **Felder**) dienen zur Speicherung größerer Datenmengen gleichen Datentyps. Ein **eindimensionales Array** lässt sich wie folgt definieren:

```
Typname  Arrayname [Anzahl];
```

Wobei **Anzahl** ein konstanter positiver Ganzzahl-Ausdruck sein muss.

```
int    alpha[5];    // Array aus 5 Elementen vom Typ int
char   beta[6];     // Array aus 6 Elementen vom Typ char
```

- Die Elemente eines Arrays können über den **Array-Index** angesprochen werden. Die Indizes beginnen stets mit 0. Die Elemente des Arrays **alpha** sind also:

```
alpha[0], alpha[1], alpha[2], alpha[3], alpha[4].
```

**Achtung:** Das n-te Element des Arrays **alpha** ist **alpha[n-1]**, (nicht **alpha[n]**).

- Es gibt *keine Fehlermeldung*, wenn zur Laufzeit des Programms der Index nicht im zulässigen Wertebereich liegt.

## Ein Beispiel

```
/* Eine Zeile einlesen und umgekehrt ausgeben */

#include <stdio.h>
#define MAXL 100

void main() {
    int i;    char puffer[MAXL];

    printf("Bitte geben Sie eine Zeile Text ein: \n\n");

    for( i = 0; i < MAXL; i++ ) {
        int c = getchar();

        if ( c == '\n' || c == EOF) break;

        puffer[i] = (char) c;           // Zeile einlesen
    }
    putchar('\n');                     // neue Zeile

    while( --i >= 0 )                  // Zeile rueckwaerts ausgeben
        putchar( (int) puffer[i] );

    putchar('\n');
}
```

## Initialisierung von Arrays

Die **Initialisierung** eines Arrays kann **manuell** wie folgt erfolgen.

```
int num[3] = {2, 4*5, 6};
```

Diese ist gleichwertig zu

```
int num[3];    num[0] = 2;    num[1] = 4*5;    num[2] = 6;
```

Ein Array kann auch ohne Angabe der Anzahl seiner Elemente initialisiert werden:

```
int num[] = {2, 4*5, 6};
```

Bei längeren Arrays spart man fehlerträchtiges Abzählen.

## Strings

Ein **String** (Zeichenkette) ist eine Zeichenfolge, die mit dem **String-Endezeichen** (**Nullzeichen**) `'\0'` abschließt. Jeder String wird in einem char-Array gespeichert. Der char-Array selbst muss mindestens 1 Byte länger als die Zeichenfolge (für `'\0'`).

Die **Initialisierung** eines char-Arrays kann durch die Angabe des Strings erfolgen:

```
char str1[]={ 'H', 'a', 'l', 'l', 'o', '\0' };
```

Äquivalent aber einfacher kann der String wie folgt initialisiert werden:

```
char str1[] = "Hallo";  
char str2[100] = "Helau";
```

(str2 hat die Länge 100, wobei nur die ersten sechs Bytes belegt.)

Eine Zuweisung einer Zeichenkette an ein Array kann nur bei der Initialisierung erfolgen. Im weiteren Programmablauf sind spezielle Bibliotheksfunktionen für diese Zwecke notwendig. Z.B. `strcpy()`, `strcat()` ...

Weitere Informationen in der C-Dokumentation unter **string.h**

(siehe Vorlesungsseite:

<http://www-ti.informatik.tu-cottbus.de/Studium/06ZgEinfProg/index.html>)

## Zeiger, Adressen und Arrays

In C sind Array-Variablen nur Zeiger auf das ersten Element des Arrays. (enthalten also die Adresse des 1. Elementes)

Z.B., für das Array `int alpha[5]`:

- `alpha` ist gleich `&alpha[0]`.
- `alpha[0]` und `*alpha` bezeichnen beide das erste Element des Arrays.
- `alpha[i]` und `*(alpha+i)` bezeichnen beide das (i+1)-te Element.

D.h., der Ausdruck `(alpha + i)` ist ein Zeiger, der auf `alpha[i]` zeigt.

Element vom Array	<code>alpha[0]</code>	<code>alpha[1]</code>	<code>alpha[2]</code>	<code>alpha[3]</code>	<code>alpha[4]</code>
Zeiger	<code>alpha</code>	<code>alpha+1</code>	<code>alpha+2</code>	<code>alpha+3</code>	<code>alpha+4</code>
Adresse (Beispiel)	2	6	10	14	18

Beispiel: `char ort[] = "Muenchen";`

`ort` (wie `&ort[0]`) bezeichnet die Adresse der Speicherzelle für das Zeichen 'M'.

## Zeigerarithmetik

Unter **Zeigerarithmetik** versteht man die Operationen, welche man mit Zeigern durchführen kann:

```
Zeiger + Ganzzahl    --> Zeiger
Zeiger - Zeiger     --> Ganzzahl
```

Ist  $p$  ein Zeiger von Typ  $T$  und  $n$  eine Ganzzahl, so ist der Ausdruck  $p + n$  wieder ein Zeiger. Er zeigt auf das  $n$ -te Objekt des Typs  $T$  nach dem Objekt, auf das  $p$  gerade zeigt. D.h.,  $p + n$  zeigt auf die Adresse:  $p + n * \text{sizeof}( T )$

(Auch die Operatoren  $++$ ,  $--$ ,  $+=$ ,  $-=$  können auf Zeiger verwendet werden.)

## Zeiger als Parameter von Funktionen

Ist bei einem Funktionsaufruf eine Variable als Argument angegeben, so erhält die Funktion nur den Wert der Variable (Call-by-value) als Kopie. Das Argument selbst kann durch die Funktion nicht geändert werden.

```
void fswap1( float x, float y ){
    float temp;
    temp = x;  x = y;  y = temp;
}
```

Der Aufruf `fswap1(a, b)` vertauscht die float Werte von  $a$  und  $b$  nicht.

Um den Wert eines Arguments durch den Funktionsaufruf zu ändern, muss Zeiger auf das Argument übergeben werden (Call-by-reference), wie bei der Funktion `scanf()`.

Die folgende Funktion `fswap2` kann die Werte von zwei Variablen vertauschen:

```
void fswap2( float *x, float *y ){
    float temp;
    temp = *x;  *x = *y;  *y = temp;
}
```

## Demonstration

```
#include <stdio.h>

void fswap1(float x, float y);    // Prototyp (siehe letzte Folie)
void fswap2(float *x, float *y); // Prototyp

void main()
{
    float  Zahl1, Zahl2;

    printf("Geben Sie zwei float Zahlen ein:\n");
    printf("Zahl1 = ");    scanf("%f", &Zahl1);
    printf("Zahl2 = ");    scanf("%f", &Zahl2);

    fswap1(Zahl1, Zahl2);
    printf("\n" "Nach dem Aufruf vom fswap1(Zahl1, Zahl2)\n"
           "Zahl1 = %g,  Zahl2 = %g\n", Zahl1, Zahl2);

    fswap2(&Zahl1, &Zahl2);
    printf("\n" "Nach dem Aufruf vom fswap2(Zahl1, Zahl2)\n"
           "Zahl1 = %g,  Zahl2 = %g\n", Zahl1, Zahl2);
}
```

## Arrays als Parameter von Funktionen

Ein Array kann als Parameter an eine Funktion übergeben werden. Dabei stellt der Arrayname einen Zeiger auf das erste Element des Arrays dar. Zum Beispiel:

```
float  Zahl[2] = {1, 2};

fswap1(Zahl[0], Zahl[1]);
printf("Zahl1 = %g,  Zahl2 = %g\n", Zahl[0], Zahl[1]);

fswap2(&Zahl, &Zahl[1]);
printf("Zahl1 = %g,  Zahl2 = %g\n", Zahl[0], Zahl[1]);
```

Erhält die Funktion `puts()` als Argument die Adresse eines Elementes von einem String, werden ab diese Adresse die Zeichen des Strings bis Stringendezeichen ausgegeben.

```
char ort[] = "Muenchen";
puts( ort );           // "Muenchen" ausgeben
puts( &ort[0] );      // "Muenchen" ausgeben
puts( &ort[3] );      // "nchen" ausgeben
```

## Mehrdimensionale Arrays

Ein  $(n+1)$ -dimensionales Array ist ein eindimensionales Array, dessen Komponenten  $n$ -dimensionale Arrays sind. Z. B.

```
int alpha[3][4];
```

deklariert ein zwei-dimensionale Array **alpha** mit 3 Zeile und 4 Spalten:

alpha[0]	alpha[0][0]	alpha[0][1]	alpha[0][2]	alpha[0][3]
alpha[1]	alpha[1][0]	alpha[1][1]	alpha[1][2]	alpha[1][3]
alpha[2]	alpha[2][0]	alpha[2][1]	alpha[2][2]	alpha[2][3]

Jedes der 12 (= 3\*4) Elemente ist vom Typ **int** und kann einzeln angesprochen werden. Z.B., mit der Zuweisung

```
alpha[1][3] = 6;
```

bekommt das Element in der Zeile 2 und Spalte 4 den Wert 6.

Ein mehrdimensionale Array kann bereits bei seiner Definition **initialisiert** werden:

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

## Beispiel von Mehrdimensionale Arrays

```
#include <stdio.h>  
char vertreter[2][20] = { "Lauber, Otto",  
                          "Forsch, Heidi" };  
/*Jeder Vertreter hat vier verschiedene Artikel im Sortiment zu verkaufen */  
char artikel_name[4][20] = {"A", "B", "C", "D"};  
int artikel_anz[2][4] = { {20, 5, 30, 17},  
                          {150, 120, 90, 110} };  
  
void main() {  
    for (int i = 0; i < 2; ++i)  
    {  
        printf("\n\n");  
        printf("Vertreter: %s\n", vertreter[i]);  
  
        printf("Verkaufte Artikel:\t");  
        for(int j = 0; j < 4; ++j)  
            printf("%8s", artikel_name[j]);  
  
        printf("\n" "Verkaufte Stueckzahlen:\t");  
        for(int j = 0; j < 4; ++j)  
            printf("%8d", artikel_anz[i][j]);  
    }  
}
```

# Strukturen

Ein **Datensatz** fasst verschiedene logisch zusammengehörende Daten zur einer Einheit zusammen. Z.B., eine Adresse soll aus folgende Informationen bestehen:

Strasse; Ort; Telefon;

Alle Informationen über **Adresse** lassen sich in C in einer **Struktur** darstellen. Ihre einzelnen Teile heißen **Komponente** der Struktur.

Ein **Strukturtyp** ist ein **Benutzer-definierter zusammengesetzter Datentyp**, der aus einer festen Anzahl vom **Komponenten** besteht:

```
struct Name {
    komponententyp1 komponente1;
    komponententyp2 komponente2;
    . . . .
    komponententyp(n) komponente(n);
};
```

Der hier selbst definierte Typname ist **struct Name**. Mit dem Strukturtyp können nun Variablen definiert werden:

```
struct Name a, b;
```

## Definition von Strukturen

Beispiel für Typdefinitionen von Strukturen:

```
struct Adresse {
    char strasse [30];
    char ort [30];
    unsigned long tel;
};

struct Student {
    int matrikelnummer;
    char name [20];
    struct Adresse wohnort;
};
```

Beispiel für die Definition von Strukturvariablen:

```
struct Student Meyer, Mueller;
struct Student InfoMaster [50]; // eine Array aus 50 Studenten
```

- Die Strukturvariable **Meyer** hat drei Komponenten.
- Komponenten können beliebigen Typ (auch Strukturtyp) sein.
- Alle Komponenten-Namen müssen verschieden sein.

## Initialisierung von Strukturvariablen

Eine Strukturvariable kann bei der Definition **initialisiert** werden. Z.B.

```
struct Student Meyer = { 1234567,
                          David Meyer,
                          { Ewald-Halse-Str.,
                            Cottbus,
                            0355696969,
                          };
};
```

### Zugriff auf Komponenten

Die Komponenten einer Struktur werden mit dem Punktoperator ( . ) ausgewählt:

```
Meyer.name           // David Meyer
Meyer.wohnort.tel    // 0355696969
```

Die Komponenten können wie folgt geändert werden:

```
Meyer.matrikelnummer = 7654321;
strcpy(Meyer.wohnort.strasse, "Bahnhofstrasse");
strcpy(Meyer.wohnort.ort, "Berlin");
```

## Zeiger auf Strukturen

**Strukturvariablen** können als Argumente an eine Funktion übergeben werden. In diesem Fall erhält die Funktion eine Kopie der Struktur (Call-by-Value).

Um auch den Inhalt der Struktur-Komponenten zu ändern, muss der Zeiger auf die Strukturvariable übergeben werden. Zeiger auf Strukturvariablen werden wie normale Zeiger vereinbart und behandelt.

```
struct Student stdt1, *stdt2;
stdt2 = &stdt1;
(*stdt2).matrikelnummer = 1234567;
```

Ein Struktur-Zeiger kann mit dem Pfeiloperator ( -> ) auf Komponenten zugreifen.

```
stdt2->matrikelnummer // gleich wie (*stdt2).matrikelnummer
&(stdt2->matrikelnummer) // gleich wie &((*stdt2).matrikelnummer)
```

## Aufzählungstypen

Ein **Aufzählungstyp** ist ein Datentyp, dessen Wertebereich nur bestimmte ganze Zahlen umfasst. Die möglichen Werte und deren Namen werden durch eine Aufzählung festgelegt.

```
enum Farbe { ROT, GELB, GRUEN };
```

- Der Aufzählungstyp hat den Namen **enum Farbe**;
- Die in der Liste angegebenen Namen ( ROT, GELB, GRUEN) bezeichnen drei Konstanten vom Typ **int**;
- Die Konstanten **ROT, GELB, GRUEN** besetzen die Werte 0, 1 und 2; Die erste Konstante hat den Wert 0, jede weitere Konstante hat einen um 1 höheren Wert als der Vorgänger.

- Die Werte der Konstanten in der Liste können durch explizit Wertzuweisung beeinflusst werden:

```
enum grundfarbe { blau = 1, gruen = 2, rot = 4} farbe;
```

- Die angegebene Werte dürfen beliebige int-Konstanten sein. Verschiedene Konstanten können denselben Wert erhalten. Ihre Name muss verschieden von Variablennamen mit derselben Sichtbarkeit.

```
enum schalter { aus, off = 0, ein, on = 1}
```

## Beispiel (1/2)

```
#include <stdio.h>
#include <time.h>

enum farben { rot, gelb, gruen };

void warte( int sekunden);
void schalte( int farbnr);

void main() {
    enum farben phase = rot;
    printf("\n\n");

    while(1){
        schalte( phase);
        switch( phase)
        {
            case rot:    warte( 23);    phase = gelb;    break;
            case gelb:   warte( 3);     phase = gruen;   break;
            case gruen:  warte( 20);    phase = rot;     break;
            default:    phase = rot;    break;
        }
    }
}
```

(nächste Seite weiter)

## Beispiel (2/2)

```
void warte( int sek)
{
    long start = time(NULL);
    while( time(NULL) < start + sek) { }
}
```

```
void schalte( int farbnr)
{
    static char *kontroll_str[] = { "ROT", "GELB", "GRUEN\n" };
    printf("\t %s\n", kontroll_str[farbnr] );
}
```