Teil 5: Felder und Zeichenketten

Gliederung

Felder (Arrays)

Funktionsparameter

Zeichenketten

Mehrdimensionale Felder

Felder (Arrays)

Felder (Arrays)

- Verbundtypen
- variable Anzahl von **Elementen** eines Datentyps
- sequenzielle Ablage im Speicher
- ein- oder mehrdimensional
- Anwendung: Vektoren,
  Matrizen,
  Tabellen,
  Zeichenketten (**Strings**)
- Definition

```
datentyp bezeichner[anzahl];
```

Beispiele

```
int zahlen[40];           // Array vom Typ Integer mit 40
                          // einzelnen Integer, 40 * 4 Byte = 160 Byte
char satz[20];            // String mit Platz für 19 Buchstaben
                          // + '\0' Byte = 80 Byte
float kommazahlen[10];    // Feld mit 10 floats
```

Ein "String" ist ein Array vom Type **char**.

```
char string[80 + 1];      // Bietet "Platz" für 80 Zeichen
```

Definition von Feldern

- Größe des Feldes muss immer exakt angegeben werden (keine Variable als Größe)
- Größe muss eine Ganzzahl sein

```
int x = 5;
float zahlen[x];        // FALSCH
float zahlen[10];       // RICHTIG
```

- Zugriff auf einen Index kann jedoch über Variablen erfolgen

```
zahlen[x] = 2.34;       // RICHTIG
```

Initialisierung von Feldern

- Möglichkeiten der Initialisierung:

```
int zahlen[5] = {0};                // Auto-Initialisierung
                                    // setzt alle Elemente auf 0
int zahlen[5] = {0, 1, 2, 3, 4};    // Voll-Initialisierung
int zahlen[] = {0, 1, 2, 3, 4};     // Compiler zählt selbst
```

- Strings:

```
char wochentag[] = "Samstag";
```

ist gleichbedeutend mit

```
char wochentag[] = {'S','a','m','s','t','a','g','\0'};
```

oder auch

```
char wochentag[8];
wochentag[0] = 'S';
wochentag[1] = 'a';  // usw.
```

Besonderheiten von Feldern

- Vergleiche zwischen Arrays `if (feld1 == feld2)` sind **nicht** möglich!
- Zuweisungen der Form `feld2 = feld1;` sind ebenfalls **nicht** möglich!

  → Lösung: elementweise vergleichen/kopieren oder gesamten Speicherbereich kopieren

- Gültige Indizes liegen im Bereich **0** bis **anzahl-1**:

```
feld[0] = 123;
...
feld[9] = 456;
feld[10] = 789;
```

- Variable Feldgrenzen sind ab C99 für lokale Variablen möglich:

```
{
 int x = 7;
 int feld[x];
 ...
}
```

Felder und Schleifen

```
#include <stdio.h>

int main()
{
    int i;
    int array[100];

    for (i = 0; i <= 99; i++)
    {
        printf("%d\n", array[i]);
    }

    return 0;
}
```

Aufgabe 1

```
/* Feld mit den natuerlichen Zahlen von 1 bis MAX belegen
   und ausgeben */

#define MAX 100

int i,
    j = 0,
    feld[MAX];

int main()
{

  ...


  return 0;
}
```

Aufgabe 2

```
/* Feld mit der Summe der natuerlichen Zahlen von 1 bis MAX
   belegen und ausgeben */

#define MAX 100

int i,
    j = 0,
    feld[MAX];

int main()
{

  ...


  return 0;
}
```

Felder kopieren

- Da eine direkte Zuweisung `array1 = array2;` nicht möglich ist, muss dies mit Hilfe einer Schleife erfolgen

```
#include <stdio.h>

int main()
{
    int i;
    int array1[100];
    int array2[100];


    for (i=0; i <= 99; i++)
    {
       array2[i] = array1[i];
    }

    return 0;
}
```

Speicherabbild

```
int i, alpha[5];
```

alpha[0] int	alpha[1] int	alpha[2] int	alpha[3] int	alpha[4] int
?	?	?	?	?

```
for (i = 0; i < 5; i++)
    alpha[i] = i * i;
```

alpha[0] int	alpha[1] int	alpha[2] int	alpha[3] int	alpha[4] int
0	1	4	9	16

Felder als Parameter

```
int main()
{
  int i;
  int einFeld[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

  Ausgabe(einFeld, 10);
  return 0;
}
```

Felder als Parameter

```
int main()
{
  int i;
  int einFeld[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

  Ausgabe(einFeld, 10);
  return 0;
}


void Ausgabe(int f[], int laenge)
{
  int i;

  for (i = 0; i < laenge; i++)
    printf("%d\n", f[i]);
}
```

Felder als Parameter

```
void Ausgabe(int[], int);


int main()
{
  int i;
  int einFeld[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

  Ausgabe(einFeld, 10);
  return 0;
}


void Ausgabe(int f[], int laenge)
{
  int i;

  for (i = 0; i < laenge; i++)
    printf("%d\n", f[i]);
}
```

call by reference

```
void erhoeheUmEins(int feld[], int laenge)
{
    int i;

    for (i = 0; i < laenge; i++)
        feld[i] = feld[i] + 1;
}
```

call by reference

```
void erhoeheUmEins(int feld[], int laenge)
{
    int i;

    for (i = 0; i < laenge; i++)
        feld[i] = feld[i] + 1;
}

int main()
{
    int einFeld[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    erhoeheUmEins(einFeld, 10);

    return 0;
}
```

call by reference

```
void erhoeheUmEins(int feld[], int laenge)
{
    int i;

    for (i = 0; i < laenge; i++)
        feld[i] = feld[i] + 1;
}

int main()
{
    int einFeld[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    Ausgabe(einFeld, 10);
    erhoeheUmEins(einFeld, 10);
    Ausgabe(einFeld, 10);

    return 0;
}
```

Bibliotheksfunktionen in <string.h>

- Länge eines Strings ermitteln

```
a = strlen(s1);   // liefert die Länge, ohne das '\0'-Byte
```

- String kopieren

```
strcpy(ziel_string, quell_string);
```

- String an einen anderen String anhängen

```
strcat(ziel_string, quell_string);
```

- Strings vergleichen

```
int a;

a = strcmp(s1, s2)

-1 wenn   s1 < s
 0 wenn   s1 == s2
 1 wenn   s1 > s2
```

Strings kopieren: strcpy()

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

int main()
{
    char ziel[81], quelle[81];

   // kopiert "Ein kleiner Text" in quelle
   strcpy(quelle, "Ein kleiner Text");

   // kopiert den Inhalt von quelle in ziel
   strcpy(ziel, quelle);

   printf("%s\n", ziel);

   return 0;
}
```

Strings anhängen: strcat()

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

int main()
{
    char ziel[81], quelle[81];

   strcpy(quelle, "Teil 2");
   strcpy(ziel, "Teil 1 ");

   // haengt den Inhalt von quelle an ziel an
   strcat(ziel, quelle);

   printf("%s\n", ziel);

   return 0;
}
```

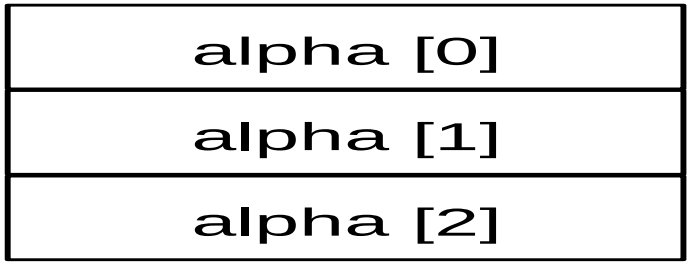
Strings vergleichen: strcmp()

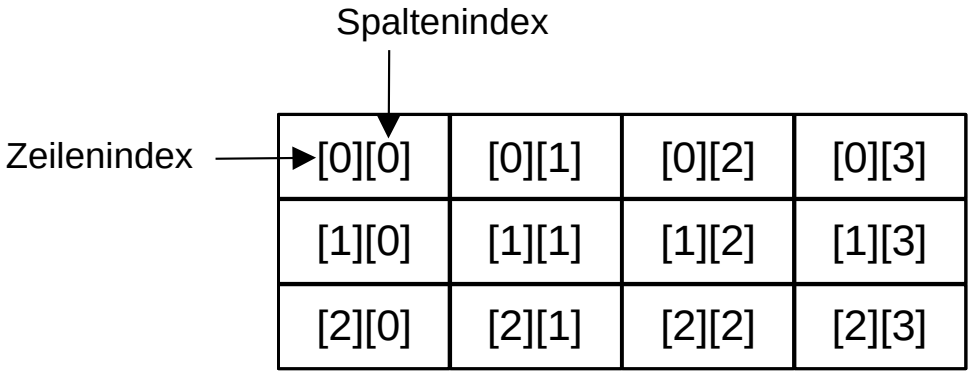
```
#include <stdio.h>
#include <string.h>

int main()
{
   char ziel[81], quelle[81];
   int ergebnis;

   strcpy(string1, "Test");
   strcpy(string2, "Test");

   // vergleicht den Inhalt von string1 und string2
   ergebnis = strcmp(string1, string2);

   if (ergebnis == 0)
      printf("Beide Strings sind gleich.\n");

   printf("%s\n", ziel);

   return 0;
}
```

Mehrdimensionale Felder

Definition

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





Allgemein kann ein n-dimensionales Array als ein eindimensionales Array, dessen Komponenten (n-1)-dimensionale Arrays sind, interpretiert werden.

Initialisierung

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

1	3	5	7
2	4	6	8
3	5	7	9

Beispiel

```
/* Alle Elemente im Feld alpha[][] auf 0 setzen */


#include <stdio.h>

int main()
{
   int alpha[3][4];
   int spalte, zeile, auswahl;

   for (zeile = 0; zeile < 3; zeile++)
   {
       for (spalte=0; spalte < 10; spalte++)
       {
           alpha[zeile][spalte] = 0;
       }
   }

   return 0;
}
```

Speicherabbild

- für **`char`** `array[3][4];` werden im Speicher 3 * 4 = 12 Byte reserviert
- In `array` ist die Anfangsadresse des Arrays gespeichert: `array[0][0]`
- Die anderen Zellen werden nach dieser Formel berechnet:

```
Adresse eines Speicherelements =
Anfangsadresse + ( Spaltennummer *
Zeilenanzahl + Zeilennummer )
```

- Beispiel:

```
char array[3][4] an Adresse 12FF80

array[2][1]: 12FF80 + ( 2 * 4 + 1) = 12FF89
array[0][3]: 12FF80 + ( 0 * 4 + 3) = 12FF83
array[1][0]: 12FF80 + ( 1 * 4 + 0) = 12FF84
```

	...	....
array[0][0]	12FF80	00000000
array[0][1]	12FF81	00000000
array[0][2]	12FF82	00000000
array[0][3]	12FF83	00000000
array[1][0]	12FF84	00000000
array[1][1]	12FF85	00000000
array[1][2]	12FF86	00000000
array[1][3]	12FF87	00000000
array[2][0]	12FF88	00000000
array[2][1]	12FF89	00000000
array[2][2]	12FF90	00000000
array[2][3]	12FF91	00000000
	...	...

Felder (mehrdimensional) als Parameter

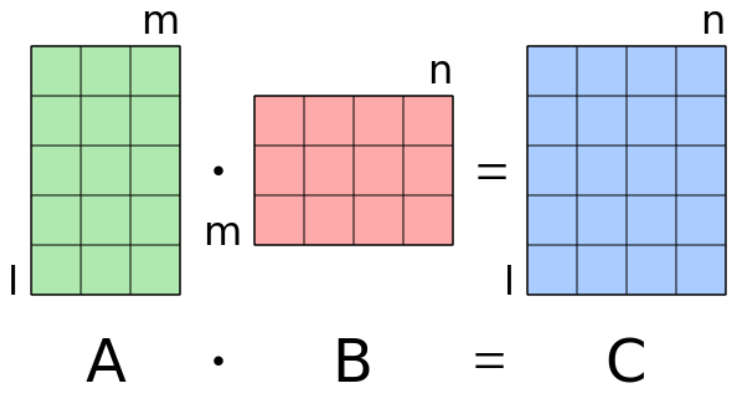
```
void AusgabeMatrix(_____________________________________);

int main()
{
  int i;
  int matrix[3][4] = {
                        { 1,  2,  3,  4 },
                        { 5,  6,  7,  8 },
                        { 9, 10, 11, 12 }
                      };

  AusgabeMatrix( matrix, 3);
  return 0;
}

void AusgabeMatrix( int matrix[][4], int zeilen )
{
  int i, j;
  for ( i = 0; i < zeilen; i++ )
    for ( j = 0; j < 4; j++)
       printf("%d ", matrix[i][j];
}
```

Fallbeispiel: Matrizenmultiplikation

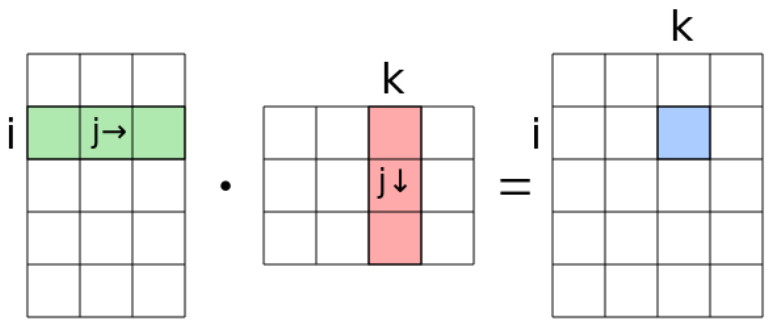
Problemstellung: Programm zur Berechnung des Produkts zweier Matrizen

Voraussetzung:

$A_{Spaltenzahl} = B_{Zeilenzahl}$



$$c_{ik} = \sum_{j=1}^{m} a_{ij} \cdot b_{jk}$$



```
#define MAX 10

int main()
{
    int zA, sA, zB, sB;
    float A[MAX][MAX], B[MAX][MAX], C[MAX][MAX];

    printf("\nMatrix a einlesen: \n");
    printf("Zahl der Zeilen : "); scanf("%d", &zA);
    printf("Zahl der Spalten: "); scanf("%d", &sA);
    mat_lesen(A, zA, sA);

    printf("\nMatrix b einlesen: \n");
    printf("Zahl der Zeilen : "); scanf("%d", &zB);
    printf("Zahl der Spalten: "); scanf("%d", &sB);
    mat_lesen(B, zB, sB);

    mat_mult(A, B,  ...   );

    printf("\nDie Ergebnismatrix c lautet: \n\n");
    mat_drucken(C, zA, sB);

    return 0;
}
```