Arrays

Einführung in die Programmierung

Michael Felderer (QE)

Institut für Informatik, Universität Innsbruck

Arrays (Vektoren)

- Ein Array ist die Zusammenfassung von mehreren Variablen des gleichen Typs unter einem gemeinsamen Typ.
- Allgemeine Form
  - **`Typname Arrayname [GROESSE]`**
- Beispiele

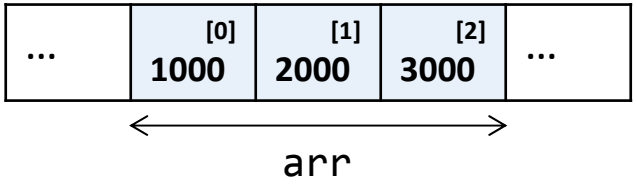
```
int counter[5];
char letters[10];
```

- GROESSE ist eine positive ganze Zahl.
  - Konstante oder konstanter Integer-Ausdruck
  - Zur Laufzeit reservierter Speicher mit Hilfe der Funktion `malloc` oder `calloc` wird im Kapitel über dynamische Speicherverwaltung besprochen!
- Array mit n Elementen
  - Indizierung beginnt bei 0 und endet bei n-1.
  - Beispiel: `counter[1] = 3; // 2. Element auf 3 setzen`

Beispiel (Array)

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX 10

int main(void) {
     int arr[3];
     arr[0] = 1000;
     arr[1] = 2000;
     arr[2] = 3000;
     printf("arr[0] = %d\n", arr[0]);
     printf("arr[1] = %d\n", arr[1]);
     printf("arr[2] = %d\n", arr[2]);
     int arr2[MAX];
     for (int i = 0; i < MAX; i++)
          arr2[i] = i * i;
     for (int i = 0; i < MAX; i++)
          printf("arr2[%d] = %d\n", i, arr2[i]);
     return EXIT_SUCCESS;
}
```



Ausgabe:

```
arr[0] = 1000
arr[1] = 2000
arr[2] = 3000
arr2[0] = 0
arr2[1] = 1
arr2[2] = 4
arr2[3] = 9
arr2[4] = 16
arr2[5] = 25
arr2[6] = 36
arr2[7] = 49
arr2[8] = 64
arr2[9] = 81
```

Array – Überlauf

- **Achtung!**
  - Beim Überschreiten (oder Unterschreiten) des zulässigen Indexbereiches werden **keine** Übersetzungsfehler erzeugt.
- Programm greift auf Speicherbereiche vor bzw. nach dem Array zu.
  - Dort befinden sich aber andere Daten!
- **Das Programm kann abstürzen oder auch weiterlaufen.**
  - **Man hat aber irgendwelche Daten verändert bzw. gelesen, auf die man nicht zugreifen wollte!**
  - **Auf unterschiedlichen Architekturen kann es zu unterschiedlichem Verhalten kommen!**

Beispiel (Arrayüberlauf am zid-gpl)

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
     const int MAX = 5;
     float farr[3];
     farr[0] = 1000.0;
     farr[1] = 2000.0;
     farr[2] = 3000.0;
     printf("farr[0] = %f\n", farr[0]);
     printf("farr[1] = %f\n", farr[1]);
     printf("farr[2] = %f\n", farr[2]);
     int iarr[MAX];
     for (int i = 0; i <= MAX + 12; i++)
          iarr[i] = i * i;
     for (int i = 0; i <= MAX + 12; i++)
          printf("iarr[%d] = %d\n", i, iarr[i]);
     printf("farr[0] = %f\n", farr[0]);
     printf("farr[1] = %f\n", farr[1]);
     printf("farr[2] = %f\n", farr[2]);
     return EXIT_SUCCESS;
}
```

Ausgabe:

```
farr[0] = 1000.000000
farr[1] = 2000.000000
farr[2] = 3000.000000
iarr[0] = 0
iarr[1] = 1
iarr[2] = 4
iarr[3] = 9
iarr[4] = 16
iarr[5] = 25
iarr[6] = 36
iarr[7] = 49
iarr[8] = 64
iarr[9] = 81
iarr[10] = 100
iarr[11] = 121
iarr[12] = 144
iarr[13] = 169
iarr[14] = 196
iarr[15] = 225
iarr[16] = 256
iarr[17] = 289
farr[0] = 0.000000
farr[1] = 0.000000
farr[2] = 3000.000000
```

Überlauf

Initialisierung von Arrays

- Arrays können bereits bei der Definition mit einer Initialisierungsliste explizit initialisiert werden:

```
float farr[5] = {0.75, 1.0, 2.5, 5.0, 7.5};
```

- Die Längenangabe kann bei der Initialisierung weggelassen werden:

```
float farr2[] = {0.75, 1.0, 2.5, 5.5, 6.0};
```

- Es müssen auch nicht alle Elemente angegeben werden (die restlichen Elemente werden mit 0 initialisiert):

```
int iarr[5] = {100, 200};
```

- Lokale Arrays kann man vollständig mit 0 auf folgende Art und Weise initialisieren:

```
int iarr2[5] = {0};
```

- Bestimmte Elemente können auch ausgewählt werden (C99):

```
int iarr3[5] = {100, [4] = 500};
```

Beispiel (Initialisierung)

```c
#include <stdio.h>
#include <stdlib.h>

int main (void) {
	const int MAX = 5;
	float farr[5] = {0.75, 1.0, 2.5, 5.0, 7.5};
	float farr2[] = {0.75, 1.0, 2.5, 5.5, 6.0};
	int iarr[5] = {100, 200};
	int iarr2[5] = {0};
	int iarr3[5] = {100, [4] = 500};
	for (int i = 0; i < MAX; i ++)
		printf("%f,", farr[i]);
	printf("\n");
	for (int i = 0; i < MAX; i ++)
		printf("%f,", farr2[i]);
	printf("\n");
	for (int i = 0; i < MAX; i ++)
		printf("%d,", iarr[i]);
	printf("\n");
	for (int i = 0; i < MAX; i ++)
		printf("%d,", iarr2[i]);
	printf("\n");
	for (int i = 0; i < MAX; i ++)
		printf("%d,", iarr3[i]);
	printf("\n");
	return EXIT_SUCCESS;
}
```

Ausgabe:

```
0.750000,1.000000,2.500000,5.000000,7.500000,
0.750000,1.000000,2.500000,5.500000,6.000000,
100,200,0,0,0,
0,0,0,0,0,
100,0,0,0,500,
```

Arrays an Funktionen übergeben

- Arrays können auch an Funktionen übergeben werden.
- Es wird aber **nicht** das Array übergeben (kopiert), sondern nur mehr die Anfangsadresse des Arrays.
  - Wird im Kapitel über Zeiger noch ausführlich besprochen.
- **Änderungen an einem Array in einer Funktion sind auch außerhalb der Funktion sichtbar!**
- Will man keine Änderungen haben
  - `const` verwenden!
  - Mit `const` werden alle Elemente eines Arrays als Konstanten angesehen.
    - Kann auch sonst immer bei Definitionen verwendet werden, wenn die Elemente eines Arrays nicht mehr verändert werden sollen!

Beispiel (Arrays als Parameter)

```c
#include <stdio.h>
#include <stdlib.h>

void init_array(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = i + i;
    }
}

void print_array(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("[%d] = %d\n", i, arr[i]);
    }
}

int main(void) {
    const int MAX = 3;
    int iArr[MAX];
    init_array(iArr, MAX);
    print_array(iArr, MAX);
    return EXIT_SUCCESS;
}
```

Hier könnte auch `const int arr[]` stehen.

Ausgabe:

```
[0] = 0
[1] = 2
[2] = 4
```

Zusammengesetzte Literale (Compound Literals in C99)

- Seit C99 kann man auch Literale verwenden, die den Inhalt eines Arrays darstellen.
- Folgende Aufrufe sind für die vorherige Funktion `print_array` zulässig:

```
print_array((int[]){4,5,6}, MAX);
print_array((int[3]){4,5,6}, MAX);
```

- Die übergebenen Arrays haben keine Bezeichner!

Arrays vergleichen

- Zwei Arrays sollten **nicht** mit dem Operator == verglichen werden!
  - Damit werden nur die Speicheradressen und nicht der Inhalt der Arrays verglichen!
- Ein richtiger Vergleich muss alle Elemente direkt überprüfen.
  - Einfache Variante
    - Mit Hilfe einer Schleife alle Elemente auf Gleichheit prüfen.
  - Schnellere Variante
    - Spezielle Funktion `memcmp()` verwenden (wird noch besprochen).

Beispiel (Vergleiche)

```c
#include <stdio.h>
#include <stdlib.h>
const int MAX = 5;

int check_elements(int array1[], int array2[]) {
      int diff=0;
      for (int i = 0; i < MAX; i++)
            if (array1[i] != array2[i])
                  diff++;
      return diff;
}

int main(void) {
      int array1[MAX] = { 1, 2, 3, 4, 5 };
      int array2[MAX] = { 5, 4, 3, 2, 1 };
      int array3[MAX] = { 1, 2, 3, 4, 5 };
      array1 == array1 ? printf("Equal!\n") : printf("Different!\n");
      array1 == array2 ? printf("Equal!\n") : printf("Different!\n");
      array1 == array3 ? printf("Equal!\n") : printf("Different!\n");
      check_elements(array1,array1) == 0 ? printf("Equal!\n") : printf("Different!\n");
      check_elements(array1,array2) == 0 ? printf("Equal!\n") : printf("Different!\n");
      check_elements(array1,array3) == 0 ? printf("Equal!\n") : printf("Different!\n");
      return EXIT_SUCCESS;
}
```

Ausgabe:

```
Equal!
Different!
Different!
Equal!
Different!
Equal!
```

Größe ermitteln

- Die Größe eines Arrays (Anzahl der Element) kann auch mit Hilfe des `sizeof`-Operators ermittelt werden.
  - Auf ein Array angewendet ergibt `sizeof` nur die Größe in Bytes!
  - Daher muss noch einmal durch die Größe des jeweiligen Datentyps dividiert werden.
- Beispiel (Ausgabe: `Anz Elemente : 17`):

```
int numbers[] = {3,6,3,5,6,3,8,9,4,2,7,8,9,1,2,4,5};
printf("Anz. Elemente : %zu\n", sizeof(numbers) / sizeof(int));
```

Mehrdimensionale Arrays

- In C gibt es auch mehrdimensionale Arrays.

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

  - `alpha` ist ein zweidimensionales Array mit 3 Zeilen und 4 Spalten.
  - Zuweisung: `alpha[1][1] = 5;`

- Initialisierung bei mehrdimensionalen Arrays:

```
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

Schematische Darstellung

```
int alpha[3][4] = { {1}, { 2, 4} };
```

1	0	0	0
2	4	0	0
0	0	0	0

Beispiel (Mehrdimensionales Array)

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int mdarray[2][3];
    mdarray[0][0] = 12;
    mdarray[0][1] = 23;
    mdarray[0][2] = 34;
    mdarray[1][0] = 45;
    mdarray[1][1] = 56;
    mdarray[1][2] = 67;
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            printf("[%d][%d] = %d\n", i, j, mdarray[i][j]);
        }
    }
    return EXIT_SUCCESS;
}
```

Ausgabe:

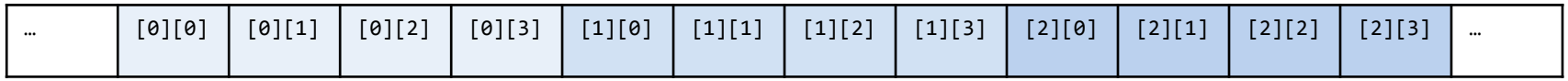
```
[0][0] = 12
[0][1] = 23
[0][2] = 34
[1][0] = 45
[1][1] = 56
[1][2] = 67
```

Mehrdimensionale Arrays an Funktionen übergeben

- Auch mehrdimensionale Arrays können an Funktionen übergeben werden.
- Auch dabei wird nicht das Array kopiert!
- Details werden später im Kapitel über Zeiger besprochen.

„Mehrdimensional“

- Die mehrdimensionalen Arrays in C sind eigentlich Arrays von Arrays.
  - Zugriff erfolgt über `[i][j][k]` und nicht über `[i, j, k]`.
  - In C wird aber von mehrdimensionalen Arrays gesprochen.
  - Andere Sprachen (z.B. Ada) unterscheiden mehrdimensionale Arrays und Arrays von Arrays explizit.
- Die mehrdimensionalen Arrays werden in C im Speicher **zeilenweise sequentiell hintereinander** abgespeichert.
  - `int arr[3][4];`



Beispiel (Matrixmultiplikation)

```c
#include <stdio.h>
#include <stdlib.h>
#define M 3
#define N 2
#define P 3

int main(void) {
     int a[M][N] = { { 1, 2 }, { 3, 4 }, { 5, 6 } };
     int b[N][P] = { { 6, 5, 4 }, { 3, 2, 1 } };
     int res[M][P] = { { 0 } };
     for (int i = 0; i < M; i++)
          for (int j = 0; j < P; j++)
               for (int k = 0; k < N; k++)
                    res[i][j] += a[i][k] * b[k][j];
     for (int i = 0; i < M; i++) {
          for (int j = 0; j < P; j++)
               printf("%2d ", res[i][j]);
          printf("\n");
     }
     return EXIT_SUCCESS;
}
```

Ausgabe:

```
12  9  6
30 23 16
48 37 26
```

Beispiel (Matrixmultiplikation – mit Funktionen)

```c
#include <stdio.h>
#include <stdlib.h>
#define M 3
#define N 2
#define P 3

void multiply_matrices(int a[M][N], int b[N][P], int c[M][P]) {
      for (int i = 0; i < M; i++)
            for (int j = 0; j < P; j++)
                  for (int k = 0; k < N; k++)
                        c[i][j] += a[i][k] * b[k][j];
}

void print_matrix(int a[M][P]) {
      for (int i = 0; i < M; i++) {
            for (int j = 0; j < P; j++)
                  printf("%2d ", a[i][j]);
            printf("\n");
      }
}

int main(void) {
      int a[M][N] = { { 1, 2 }, { 3, 4 }, { 5, 6 } };
      int b[N][P] = { { 6, 5, 4 }, { 3, 2, 1 } };
      int res[M][P] = { { 0 } };
      multiply_matrices(a, b, res);
      print_matrix(res);
      return EXIT_SUCCESS;
}
```

Ausgabe:

```
12  9  6
30 23 16
48 37 26
```

Beispiel (Probleme)

- Arraydimensionen sind fixiert.
- Man kann immer nur Arrays mit den entsprechenden Dimensionen übergeben.
- Erste Dimension kann theoretisch auch leer bleiben, d.h. der Funktionskopf von `multiply_matrices` kann auch folgendermaßen geschrieben werden:

```
void multiply_matrices(int a[][N], int b[][P], int c[][P]) { … }
```

  - Es kann immer nur die erste Dimensionsangabe weggelassen werden, alle weiteren Dimensionsangaben müssen vorhanden sein!
  - Hintergrund wird noch erklärt (Kapitel über Zeiger)!

Variable-Length Arrays (VLAs) in C99

- Die Arraydimensionen können in C99 auch mit Hilfe von Variablen spezifiziert werden.
- Beispiel

```
int a = 4;
int b = 5;
…
double arr[a][b];
```

- Auch bei Funktionen

```
int sum(int rows, int columns, int arr[rows][columns]) {
    int sum = 0;
    for(int r = 0; r < rows; r++)
            for(int c= 0; c < columns; c++)
                    sum += arr[r][c];
    return sum;
}
```

rows und **column** müssen hier vor dem Array angegeben werden!

Zeichenketten (Strings) in C

- In C werden Arrays vom Datentyp `char` zum Speichern von Zeichenketten (Strings) verwendet.
- Der C-Compiler markiert das Ende eines solchen char-Arrays mit dem Null-Zeichen `\0` (Byte, das nur 0-Bits enthält), damit das Ende der Zeichenkette erkannt werden kann.
- Die Länge eines solchen Arrays ist daher **um 1 größer** als die Anzahl der relevanten Zeichen!
- Der formale Parameter einer Funktion, der als eine Zeichenkette übergeben wird, kann vom Typ char[] sein.
  - Eine weitere Möglichkeit wird im Kapitel über Zeiger besprochen.
- Für die Bearbeitung von Strings gibt es viele Funktionen in der Standardbibliothek, deklariert in der Header-Datei `<string.h>`.
  - Beispiele: `strlen`, `strcpy`, `strcmp`

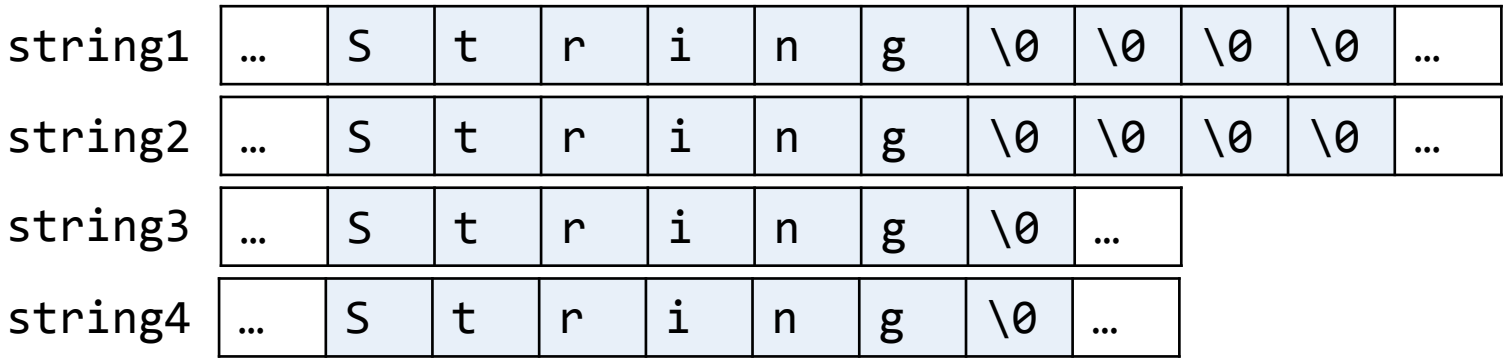
Beispiel (Strings)

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

int main(void) {
    char string1[10] = "String";
    char string2[10] = {'S', 't', 'r', 'i', 'n', 'g', '\0'};
    char string3[] = "String";
    char string4[] = {'S', 't', 'r', 'i', 'n', 'g', '\0'};
    printf("%s\n", string1);
    printf("%s\n", string2);
    printf("%s\n", string3);
    printf("%s with length %d\n", string4, strlen(string4));
    return EXIT_SUCCESS;
}
```

Ausgabe:

```
String
String
String
String with length 6
```



Null-Zeichen

- Die meisten String-Funktionen in der Standardbibliothek benutzen das Null-Zeichen in Abbruchbedingungen.
- Fehlt das Null-Zeichen, dann wird bis zum nächsten Null-Zeichen im Speicher weitergesucht.
  - Das kann manchmal sehr lang dauern und unter Umständen zu einem Absturz des Programms führen!
- Keine Größenangabe bei Deklaration (`char string[] = …`)
  - Einfache Schreibweise
  - Compiler errechnet die Array-Länge inklusive des Null-Zeichens!