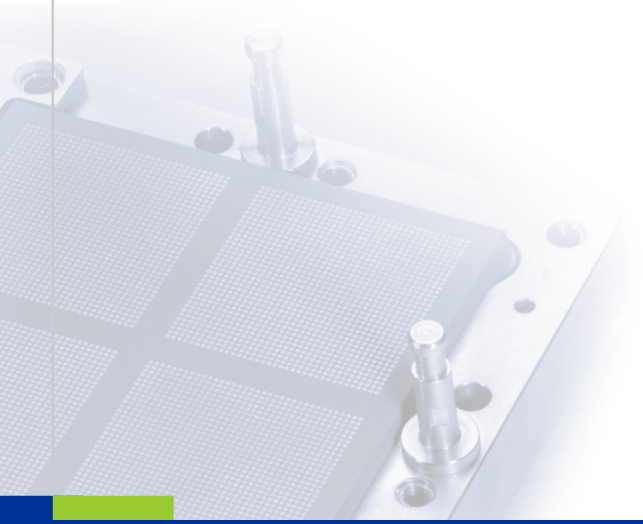
R|R|Z|N

# Programmierung mit C



## Felder; Array

# Arrays (Felder, Vektoren)



- ... sind zusammengesetzte Datenstrukturen.
- ... bestehen aus mehreren Elementen, die vom selben Datentyp sind.
- ... bestehen aus mehreren Elementen, die die gleiche Speichergröße haben und im Speicher linear abgelegt sind.
- ... können ein- oder mehrdimensional sein.
- können
  - ... Ganzzahlen sowohl als auch Fließkommazahlen speichern.
  - ... einzelne Zeichen in ihren Feldern speichern. Die einzelnen Elemente des Arrays bilden eine Zeichenkette aus Buchstaben, Ziffern und Satzzeichen.

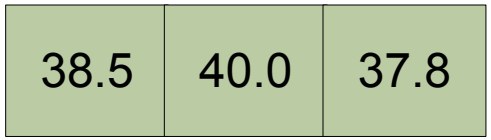
# Arrays deklarieren

R|R|Z|N

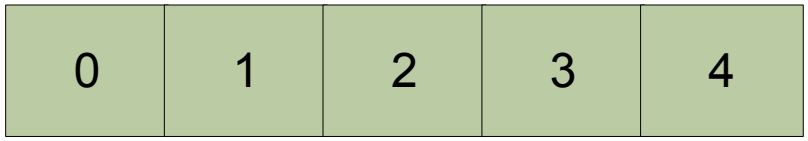
```
Datentyp Arrayname [Anzahl_der_Elemente]
```

- `Datentyp`
  - ... gibt Auskunft über den Inhalt jedes einzeln Elements in einem Array.
  - ... legt die Größe des benötigten Speichers für jedes Element fest. Die Größe des Arrays ergibt sich aus der Anzahl der Elemente mal ihren Speicherbedarf.
  - ... kann durch jeden einfachen oder benutzerdefinierten Datentyp ersetzt werden.
- `Arrayname`
  - ... ist frei wählbar, sollte aber die Nutzung widerspiegeln
  - ... ist ein Platzhalter für die Anfangsadresse des Feldes im Speicher.
- Die `Anzahl der Elemente`
  - ... eines Feldes werden immer in eckige Klammern gesetzt.
  - ... werden immer durch eine positive Ganzzahl dargestellt.
  - ... muss zum Zeitpunkt der Übersetzung bekannt sein. Der Standard C99 erlaubt auch Arrays, deren Größe erst zur Laufzeit festgelegt wird.

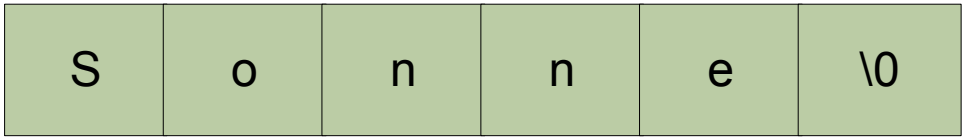
# Beispiele




```
float temperatur[3];
```




```
int zahlenreihe[5];
```




```
char wort[6];
```

# Arrays deklarieren und initialisieren

R|R|Z|N

```
Datentyp Arrayname[ ] = {Element1, Element2, ..., ElementN}
```

- In diesem Beispiel wird ein Array gleichzeitig deklariert und initialisiert werden.
- Die Werte für die einzelnen Elemente des Feldes werden durch Kommata getrennt und mit Hilfe von geschweiften Klammern zusammengefasst. Die angegebenen Werte müssen dem Datentyp des Arrays entsprechend gewählt werden.
- Mit Hilfe des Gleichheitszeichen werden den Elementen des Feldes, links vom Gleichheitszeichen, die Werte in den geschweiften Klammern zugewiesen.
- Die Anzahl der Elemente des Feldes ergibt sich aus der Anzahl der angegebenen Werte in den geschweiften Klammern. Die Angabe eines Index ist überflüssig.

# Elemente eines Feldes initialisieren

- `int Feld [10] = { 1, -3, 2 };`
  - In den eckigen Klammern wird die maximale Anzahl des Feldes angegeben. Hier hat das Array 10 Felder.
  - In den geschweiften Klammern werden ein oder mehr Initialisierungswerte für die ersten Feldelemente des Arrays angegeben.
  - Mit Hilfe des Gleichheitszeichen bekommt das erste Element den Wert 1 zugewiesen, das zweite Element den Wert -3, das dritte den Wert 2.
  - Alle anderen Feldelemente besitzen eine undefinierten Wert.
- `int Feld [10] = { 1, -3, 2 };`
  - Führt zu einer Compiler-Warnung, da die Anzahl der Initialisierungswerte die Größe des Feldes überschreitet.

# Automatische Initialisierung aller Felder

R|R|Z|N

- `int Arrayname[10] = { 0 }`
  - In den eckigen Klammern wird die maximale Anzahl des Feldes angegeben. Hier hat das Array 10 Felder.
  - In den geschweiften Klammern wird ein Initialisierungwert für jedes einzelne Feld des Arrays angegeben.
  - Mit Hilfe des Gleichheitszeichen bekommen alle 10 Felder automatisch den Wert Null zugewiesen.
- `int Feld[10] = { 1 }`
  - In den eckigen Klammern wird die maximale Anzahl des Feldes angegeben. Hier hat das Array 10 Felder.
  - In den geschweiften Klammern wird ein Initialisierungwert für das erste Feldelement des Arrays angegeben.
  - Mit Hilfe des Gleichheitszeichen bekommt das erste Feld automatisch den Wert Eins zugewiesen. Alle anderen Felder besitzen eine undefinierten Wert.
  - Eine automatisch Initialisierung aller Felder eines Arrays mit einem Wert ungleich Null ist nicht möglich!

# Einzelne Elemente eines Feldes initialisieren

R|R|Z|N

```
Arrayname[Index] = Ausdruck;
```

- Jedes Element in einem Array kann über ein Index angesprochen werden. Der Index gibt die Position eines Elementes innerhalb eines Arrays an.
- Der Index wird in eckige Klammern gesetzt und besitzt sich auf das da vorstehende Array..
- Die Elemente eines Arrays werden von 0 bis n durchnummeriert. Das erste Element hat die Position 0 in einem Array. Das letzte Element im Array hat die Position "[maximale Anzahl der Elemente] – 1".
- Falls ein Index größer als die "[maximale Anzahl der Elemente] – 1" ist, wird auf einen Speicherbereich zugegriffen, der nicht zum Array gehört. Eine Überschreitung der minimalen sowie der maximalen Grenze wird nicht vom Compiler oder Laufzeitsystem erkannt. Der Entwickler muss mögliche Fehler abfangen!

# Beispiel



```
int main(void)
{
   /* Initialisieren aller Felder mit 0 */

   int feld[10] = {0};
   float reihe[3] = {0};

   /* Zuweisung */
   reihe[0] = 1.5;
   reihe[1] = 2.5;
   reihe[2] = 3.5;
   reihe[3] = 4.5;

   return 0;
}
```

Das geht nicht!

Der Index des Feldes `reihe` darf nur von 0 bis 2 laufen!
`reihe` hat maximal drei Elemente.

# Beispiel

R|R|Z|N

```
#include<stdio.h>

#define MAX_ELEMENT 5

int main()
{
    int feld[MAX_ELEMENT];
    int count;

    for (count = 0; count < MAX_ELEMENT; count++){
        feld[count] = count * 2;
    }

    for (count = 0; count < MAX_ELEMENT; count++){
        printf("Feld %i: %i\n", count,  feld[count]);
    }

    return 0;
}
```

Die maximale Anzahl der Elemente eines Feldes wird als symbolische Konstante angelegt.

Vorteil: Die Größe eines Feldes kann für das gesamte Programm schneller angepasst werden.

# Gültigkeitsbereiche



- Ein Array kann als lokale Variable definiert werden. Die einzelnen Elemente des Array haben einen undefinierten Wert.
- Ein Array kann global definiert werden. Die einzelnen Elemente des Arrays sind automatisch mit 0 initialisiert.
- Ein Array kann statisch definiert werden. Die einzelnen Elemente des Arrays sind automatisch mit 0 initialisiert.

# Vergleichsoperatoren nutzen

R|R|Z|N

- `Arrayname[Index] == Feldname[Index]`
  - Die einzelnen Elemente eines Arrays können mit Hilfe der Vergleichsoperatoren verglichen werden.
  - Zeichenketten können nicht mit Hilfe von Vergleichsoperatoren verglichen werden. Hier muss auf Bibliotheksfunktionen zurück gegriffen werden.
- `Arrayname == Feldname`
  - Der Name eines Arrays symbolisiert die Anfangsadresse eines Feldes. In diesem Beispiel werden zwei Speicheradressen (Anfangsadressen zwei verschiedener Felder) miteinander verglichen.
  - Es können nicht alle Elemente zwei verschiedener Felder automatisch miteinander verglichen werden. Nutzen Sie für den Vergleich von allen Elementen FOR-Schleifen.

# Größe des Arrays

```
int zahlen[] = {1, 2, 3, 4, 5};
printf("Groesse: %d", sizeof(zahlen));
```

- In der ersten Zeile wird ein Array mit fünf Elementen deklariert und gleichzeitig initialisiert.
- Mit Hilfe des Operators `sizeof()` wird die Größe des Arrays in Bytes ermittelt.
  - Ein Integer-Element benötigt 4 Bytes.
  - Das Array besitzt fünf Integer-Elemente.
  - Das Array hat einen Speicherbedarf von 20 Bytes (4 * 5).

# Anzahl der Elemente ermitteln

R|R|Z|N

```
int zahlen[] = {1, 2, 3, 4, 5};
printf("Anzahl Elemente: %d", sizeof(zahlen) / sizeof(int));
```

- `sizeof(int)` ermittelt in diesem Beispiel
  - ... die Größe eines Integer-Wertes (4 Bytes) und damit den Speicherbedarf eines Elements.
  - ... sizeof(zahlen) den gesamten Speicherbedarf des Arrays (20 Bytes).
- Anschließend wird der Speicherbedarf des Feldes durch den Speicherbedarf eines Elementes geteilt.

# Ein- und Ausgabe am Bildschirm

R|R|Z|N

- `scanf("%d", &array[index])`
  - Werte für die einzelnen Elemente eines Arrays können mit Hilfe der Tastatur eingegeben werden.
  - Das Formatzeichen muss mit dem Datentyp der Feldelemente übereinstimmen.
- `printf("%d", array[index])`
  - Ein Element des Arrays wird auf dem Bildschirm ausgegeben.
- `printf("%p", array)`
  - Die Anfangsadresse des Arrays wird ausgegeben.
- Vergessen Sie nicht, die Header-Datei `<stdio.h>` für die Funktionen `scanf()` und `printf()` einzubinden.

# Mehrdimensionale Arrays

R|R|Z|N

- ... werden für die Darstellung von Matrizen oder Koordinatensystem genutzt.
- Wie viele Dimensionen maximal genutzt werden können, hängt vom verwendeten Compiler ab.
- Der benötigte Speicherplatz wächst exponentiell mit der Anzahl der Dimensionen.
  - Für ein zweidimensionales Array würde zum Beispiel der Speicherbedarf folgendermaßen berechnet:

    Größe des ersten Index * Größe des zweiten Index * sizeof(Datentyp)

- Deklaration von mehrdimensionalen Arrays:

  ```
  Datentyp arrayname[dimension1] ... [dimensionN]
  ```

- Deklaration eines zweidimensionales Arrays:

  ```
  int matrix[4][5]
  ```

  - Der erste Index (`[4]`) gibt die Anzahl der Elemente in der ersten Dimension an.
  - Der zweite Index ([5]) gibt die Anzahl der Elemente in der zweiten Dimension an.
  - Wenn man ein zweidimensionales Array als Tabelle abbildet, gibt der erste Index die Zeilen und der zweite Index die Spalten an.

- Deklaration eines dreidimensionales Arrays:

  ```
  int matrix[4][5][2]
  ```

  - Die erste Dimension entspricht der x-Achse in einem Koordinatensystem.
  - Die zweite Dimension entspricht der y-Achse in einem Koordinatensystem.
  - Die dritte Dimension entspricht der z-Achse in einem Koordinatensystem

# Zweidimensionale Arrays initialisieren

R | R | Z | N

```
int matrix [4][5] = { {1, 2, 3, 4, 5},
                      {6, 7, 8, 9, 10},
                     {11, 12, 13, 15},
                     {16, 17} };
```

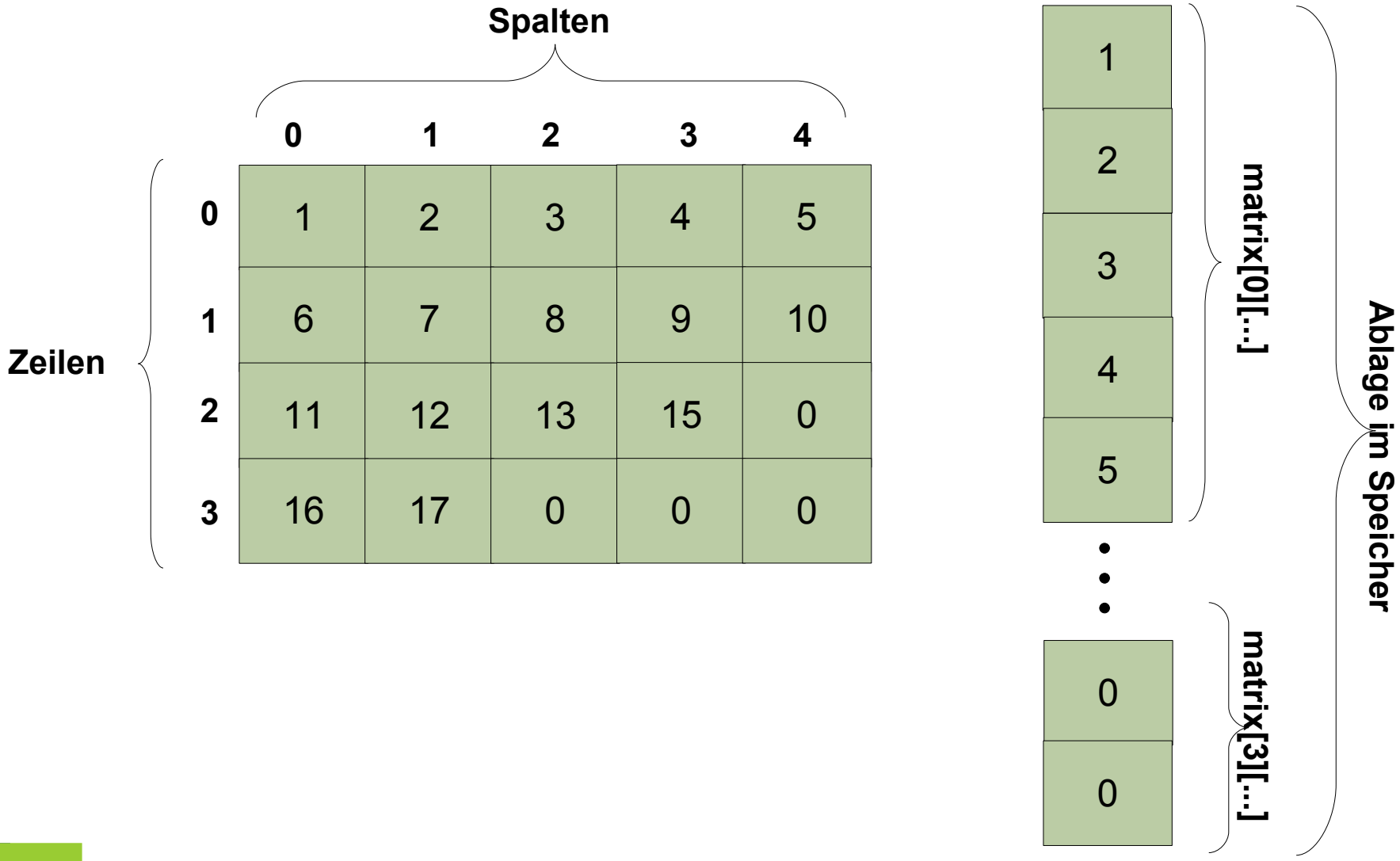
- Alle Elemente in einem Array werden mit Hilfe der geschweiften Klammern zusammengefasst.
- Innerhalb der geschweiften Klammern wird ein zweidimensionales Array wiederum zeilenweise mit Hilfe der geschweiften Klammern zusammengefasst. Die einzelnen Zeilen werden durch Kommata getrennt. In diesem Beispiel können maximal vier Zeilen initialisiert werden.
- In den geschweiften Klammern werden die Werte für die einzelnen Spalten für die jeweilige Zeile, durch Kommata getrennt angegeben. Die maximale Anzahl der Spalten richtet sich nach dem Index für die zweite Dimension. In diesem Beispiel können Werte für maximal fünf Spalten angegeben werden. Zellen, denen kein Initialisierungwert zugewiesen wird, haben den Wert Null.

# Zellen Werte zuweisen



- Der Kreuzungspunkt einer Zeile mit einer Spalte wird als Zelle bezeichnet.
- Beispiel: `matrix[2][3] = 4;`
  - Für jede Dimension wird ein Index angegeben. Die Indizes legen die Position einer Zelle in einem zweidimensionalen Raum fest. Hier wird die Zelle in der dritten Zeile und der vierten Spalte angesprochen.
  - Die Zelle an der angegebenen Position bekommt mit Hilfe des Gleichheitszeichens einen Wert zugewiesen.

# Grafische Darstellung



# Beispiel: Sonnenaufgang

R|R|Z|N

```
#include <stdio.h>
#define WOCHEN 4
#define TAGE 7
float sonnenaufgang [WOCHEN] [TAGE] = {
{6.01f, 5.57f,  5.56f,  5.54f,  5.05f,  5.51f,  5.36f},
{5.47f, 5.46f,  5.44f,  5.43f,  5.41f,  5.40f,  5.39f},
{5.37f, 5.36f,  5.34f,  5.33f,  5.32f,  5.31f,  5.30f},
{5.28f, 5.27f,  5.26f,  5.25f,  5.24f,  5.23f,  5.23f}
};

int main(void) {
    int zeile, spalte;
    printf("Tag; Mo;Di;Mi;Do;Fr;Sa;So");

    for (zeile = 0; zeile < WOCHEN; zeile++){
        printf("\nWoche%d ;", zeile);

      for (spalte=0; spalte < TAGE;  spalte++){
        printf("%.2f ;", sonnenaufgang[zeile] [spalte]);
      }
    }
```

# Beispiel: Determinate einer 2 x Matrix

R|R|Z|N

```
#include <stdio.h>
#define MAX_ELEMENT 2

int main()
{
    int matrix[MAX_ELEMENT][MAX_ELEMENT];
    int zeile;
    int spalte;
    int determinate;

    for(zeile = 0; zeile < MAX_ELEMENT; zeile++){
        for(spalte = 0; spalte < MAX_ELEMENT; spalte++){
            matrix[zeile][spalte] = (zeile + 1) * (spalte + 1);
        }
    }

    determinate = (matrix[0][0] * matrix[1][1])
    determinate = determinate - (matrix[0][1] * matrix[1][0] );
```

# Defensive Programmierung Arrays

R|R|Z|N

- Überprüfen Sie, ob die Indizes innerhalb der Grenzen des Arrays liegen.
  - Ein häufiger Fehler ist der Zugriff auf Elemente die außerhalb des Arrays liegen.
- Überprüfen Sie die Endpunkte von Arrays.
  - Greift das Programm wirklich auf das letzte Element zu, oder schon auf die Datenstruktur, die hinter dem Array liegt?
- Überprüfen Sie die richtige Reihenfolge der Indexe bei mehrdimensionalen Arrays.
  - Wie leicht wird aus `array[i][j]` versehentlich ein `array[j][i]`.
  - Indizies werden häufig nur mit einem Buchstaben gekennzeichnet. Bei mehrdimensionalen Arrays ist es aber sinnvoller, die Zeile und Spalte kenntlich zu machen.

# Strings (Zeichenketten)

R|R|Z|N

- ... sind Arrays vom Datentyp `char`.
- ... sind eine Kette von einzelnen Zeichen.
- ... enden mit dem Zeichen \0.
- ... sind meist eindimensional.
- ... können Konstanten in der Form von "Hello World" sein.

# Strings deklarieren und initialisieren

R|R|Z|N

- Deklaration eines Strings:

  ```
  char array[Anzahl_der_Elemente];
  ```

  Für ein String muss ein Array der Größe "Anzahl_der_Zeichen plus Endezeichen" angelegt werden.
- Initialisierung eines Strings:
  - Für jedes Element wird ein Zeichen angegeben.

    Jedes einzelne Zeichen wird durch Apostrophs begrenzt.

    Das Endezeichen (\0) des Strings muss manuell gesetzt werden.

    Beispiel: `char sonne[] = {'S', 'o', 'n', 'n', 'e', '\0'};`
  - Es wird eine Stringkonstante angegeben.

    Strings werden durch Anführungszeichen begrenzt.

    Jedes Element in dem Array wird automatisch mit den entsprechenden Zeichen in dem String initialisiert.

    Beispiel: `char sonne[] = {"Sonne"};`

# Werte zuweisen

R|R|Z|N

- `feld[1] = 'S'`
  - Das zweite Elemente des Feldes bekommt den Wert 'S' zugewiesen.
  - Jedem Element in einem Feld kann nur jeweils ein ASCII-Zeichen zugewiesen bekommen.
- `feld[1] = 'SA'`
  - Der Compiler gibt eine Warnung aus. Das Programm kann aber gestartet werden.
  - Häufig wird dem zweiten Element des Feldes der Wert 'S' zugewiesen.
- `feld[1] = "SA"`
  - Der Compiler zeigt einen Fehler an.
  - Der Compiler meldet, dass das Zeichen rechts vom Gleichheitszeichen nicht kompatible zu einem Char-Zeichen ist.
- `feld[1] = 1`
  - Dem zweiten Element des Feldes wird ein ASCII-Zeichen mit dem dezimalen Wert 1 zugewiesen.
  - Dem zweiten Element wird nicht die Ziffer eins zugewiesen!

# String-Endezeichen



- Ein String wird immer mit dem Endezeichen '\0' abgeschlossen.
- Alle Zeichen, die dem Endezeichen nachfolgen, werden nicht beachtet.
- Eine Stringkonstante ("Sonne") enthält ein Endezeichen. Das String-Endezeichen muss nicht gesetzt werden.

# Ausgabe am Bildschirm



- Einen String bis zum String-Endezeichen ausgeben:
  ```
  printf("String: %s", array);
  ```
  Das Endezeichen wird nie mit ausgegeben.
- Ein einzelnes Zeichen aus dem String ausgeben:
  ```
  printf("Zeichen: %c", array[Index]);
  ```

# Strings einlesen



- Einen String vom Bildschirm einlesen: `scanf("%s", array);`
  - Der Name eines Arrays symbolisiert die Anfangsadresse des Feldes im Speicher. Aus diesen Grund kann bei Stringvariablen das kaufmännische UND weggelassen werden.
  - Ein String wird nur bis zum ersten vorkommenden Leer-, Tabulator- oder Zeilen-Endezeichen eingelesen.

# Unformatierte Ein- und Ausgabe am Bildschirm



- `puts(array);` gibt ein String unformatiert am Bildschirm aus.
- `gets(array);` liest eine Zeichenkette unformatiert vom Bildschirm ein.
- Für diese Funktionen muss die Header-Datei *stdio.h* eingebunden werden.

# Eine bestimmte Anzahl von Zeichen einlesen

R|R|Z|N

```
char *fgets(char *string, int anzahl, FILE *stream)
fgets(array, 100, stdin)
```

- In dem Beispiel werden maximal 100 Zeichen von der Standardeingabe (häufig der Bildschirm) eingelesen und in dem Feld gespeichert.
- `fgets()`
  - ... setzt am Ende eines Strings automatisch ein Newline-Zeichen und ein String-Endezeichen. In dem oben genannten Beispiel werden maximal 98 beliebige Zeichen eingelesen. Die letzten zwei Elemente werden mit dem Zeichen Newline und dem String-Endezeichen belegt.
  - ... bricht automatisch nach dem Einlesen der maximalen Anzahl von Zeichen ab.
  - ... ist in der Header-Datei *stdio.h* deklariert.

# Standard-Bibliothek <string.h>



- ... enthält Funktionen, die die Arbeit mit Strings erleichern.
- ... entspricht dem ANSI-C-Standard.
- ... ist im Lieferumfang der meisten Compiler vorhanden.
- ... nutzt char-Zeiger.
  - Strings werden als Felder abgelegt.
  - Der Zeiger zeigt auf das erste Element des Arrays.

# Länge eines Strings



`size_t strlen(const char *s1);`

- .... ermittelt die Länge eines Strings.
- ... gibt die Anzahl der Zeichen wieder. Das String-Endezeichen wird nicht mit berücksichtigt.
- Der Datentyp `size_t`
  - ... steht für „size type“ (Größe des Speichers)
  - ... entspricht long unsigned int. Die Kompatibilität ist aber nicht immer implementiert.

```
#include <string.h>
int main()
{
   char sonne[] = {"Sonne"};
   size_t laenge;

   laenge = strlen(sonne);
   return 0;
}
```

# Strings anhängen

R|R|Z|N

```
char *strcat(char *ziel, const char *quelle);
```

- Der String `quelle` wird an das Ende von `ziel` angehängt.
- Das erste Element des Feldes `quelle` überschreibt das String-Endezeichen des Feldes `ziel`. Zum Schluss wird das String-Endezeichen automatisch an das Ende des neuen Strings angefügt.
- Es können nur so viele Zeichen angehängt werden, wie freie Elemente in `ziel` vorhanden ist.
  - `ziel` hat eine Länge von zehn Elementen. Das Feld belegt selber fünf Elemente.
  - Das anzuhängende Feld `quelle` hat neun belegte Elemente, die an `ziel` angehängt werden sollen.
  - Es werden mit Hilfe der Funktion `strcat()` die ersten vier belegten Felder von `quelle` an `ziel` angehängt. In das zehnte Feld von `ziel` wird das String-Endezeichen gesetzt. `quelle` kann nicht vollständig an `ziel` angehängt werden.

# Beispiel

```
char eisbaer[10] = {"Eis"};
size_t laenge;
size_t newLaenge;
int groesse;

laenge = strlen(eisbaer);
newLaenge = strlen("baer");
groesse = sizeof(eisbaer) / sizeof(char);

if (groesse >= (laenge + newLaenge + 1)){
     strcat(eisbaer, "baer");
}
```

# Eine beliebige Anzahl von Zeichen anhängen

R|R|Z|N

```
char *strncat(char *ziel, const char *quelle, size_t anzahl);
```

- Es werden `anzahl` Elemente des Strings `quelle` an `ziel` angehängt.
- `anzahl` ist immer eine Ganzzahl.
- Das String-Endezeichen wird automatisch der neuen Länge von `quelle` angepasst.
- Die Größe von ziel muss so gewählt werden, dass

  "ziel.BelegteFelder + quelle.Anzahl + String-Endezeichen" Elemente vorhanden sind.
- Fallbeispiele:
  - quelle.AnzahlElemente > anzahl

    Es wird die gewünschte Anzahl von Elemente, beginnend am Anfang des Feldes `quelle` an `ziel` angehängt.
  - quelle.AnzahlElemente < anzahl

    Das Feld `quelle` wird vollständig an `ziel` angehängt.

# Beispiel

```
char feld [MAX] = { "Hello" };
char puffer [20];
size_t laenge;

laenge = MAX - strlen(feld) + 1;

printf("Ihr Name: ");
fgets(puffer, 20, stdin);

strncat(feld, puffer, laenge);
```

`char *strcpy(char *ziel, const char *quelle);`

- `quelle` wird in `ziel` kopiert.
- Die, in `ziel` vorhandenen Elemente werden überschrieben.
- Die Größe von `ziel` wird nicht im Hinblick auf die Länge des zu kopierenden String überprüft. Falls der zu kopierende String länger ist als `ziel` ist, muss der Entwickler eine Fehlermeldung ausgeben.

```
char eis[10] = {"Eis"};
char baer[5] = {"Baer"};
size_t laenge;
int groesse;

laenge = strlen(baer);
groesse = sizeof(eis) / sizeof(char);

if (groesse >= (laenge + 1)) {
                    strcpy(eis, baer);
}
```

# n Zeichen kopieren



```
char *strncpy(char *ziel, const char *quelle, size_t anzahl);
```

- Eine bestimmte Anzahl von Zeichen aus dem String `quelle` werden in den String `ziel` kopiert.
- Die Größe des Feldes `ziel` darf nicht die Anzahl der zu kopierenden Zeichen unterschreiten.
- Es werden `anzahl` Zeichen nach `ziel` kopiert.
  - Wenn `quelle` weniger als `anzahl` Zeichen enthält, wird mit dem Ende-Zeichen aufgefüllt.
  - Wenn `quelle` mehr als `anzahl` Zeichen besitzt, wird der String `ziel` nicht mit dem Endezeichen abgeschlossen.

# Beispiel

```
#include <string.h>
##define MAX 7

int main()
{
   char blume[13] = {"Sonnenblumen"};
   char feld[MAX];

   strncpy(feld, blume, MAX - 1);
   feld[MAX - 1] = '\0';
```

# Strings vergleichen



- Strings können nicht mit Hilfe von Vergleichsoperatoren verglichen werden.
- Die Header-Datei *string.h* bietet Funktionen für den Vergleich von Strings an.
- Bei einem Vergleich von Zeichen wird der ASCII-Code des zu vergleichenden Zeichens genutzt.
- Ein Vergleich beachtet die Groß- und Kleinschreibung. Großbuchstaben liegen im Bereich 65 - 90. Kleinbuchstaben liegen im Bereich 97 – 122. Aus diesem Grund ist die Zeichenkette "HEUTE" kleiner als der String "heute". Momentan sind keine Funktionen vorhanden, die die Groß- und Kleinschreibung nicht beachten.

# Funktionen zum Vergleich von Strings

R|R|Z|N

```
int strcmp(const char *string, const char *zeichen);
int strncmp(const char *string, const char *zeichen, size_t anzahl);
```

- Strings werden zeichenweise verglichen.
- Rückgabewerte der Funktion:
  - [string] == [zeichen] = 0.
  - [string] > [zeichen] = > 0.
  - [string] < [zeichen] = < 0.

# String vollständig vergleichen

R|R|Z|N

```
#include <string.h>
int main() {
   char zeichenA[4] = {"ABC"};
   char zeichenKleinA[4] = {"abc"};
   char zeichenB[4] = {"BCD"};
   char zeichen10[10] = {"ABCDEFGIH"};
   int ret;

   ret = strcmp(zeichenA, zeichenA);        /* Rückgabewert 0 */

   ret = strcmp(zeichenA, zeichenKleinA); /* Rückgabe -1 */

   ret = strcmp(zeichenA, zeichenB);      /* Rückgabewert -1 */

   ret = strcmp(zeichenA, zeichen10);    /* Rückgabewert -1 */

   ret = strcmp(zeichenB, zeichen10);   /* Rückgabewert 1 */

   return 0;
}
```

# Teilstrings vergleichen

R|R|Z|N

```
#include <string.h>

int main() {
   char zeichen[5] = {"ABCE"};
   char reihe[5] = {"ABFG"};
   int count;
   int ret;

   for (count = strlen(zeichen); count > 0;  count--) {
      ret = strncmp(zeichen, reihe, count);

      if (ret != 0) {
            printf(" %d Zeichen sind ungleich\n", count);

      }
      else {
            printf("Gleichheit ab %d Zeichen\n", count);

            break;
      }
   }
}
```

# Ein Zeichen in Strings suchen

R|R|Z|N

```
char  *strchr(const char *string, int zeichen);
```

- Der String wird vom Anfang bis zum String-Endezeichen durchsucht.
- Die Suche wird abgebrochen, sobald das angegebene Zeichen gefunden wurde.
- Wenn die Suche erfolgreich war, wird ein Zeiger auf das angegebene Zeichen im String zurückgeliefert. Andernfalls wird ein NULL-Zeiger zurückgeliefert.
- Bei der Suche wird die Groß- und Kleinschreibung von Zeichen berücksichtigt!

# Beispiel

```
char puffer[8] = {"Eisbaer"};
int zeichen;
char *ptr;
zeichen = (int) 'e';

ptr = strrchr(puffer, zeichen);

if ( NULL == ptr) {
     printf("%c wurde nicht gefunden", zeichen);
}
else {
     printf("Index: %d", (int) (ptr - puffer));
}
```

# Teilstrings suchen

R|R|Z|N

```
char *strstr(const char *sucheIn, const char *suche);
```

- Die Funktion sucht einen Teilstring in einer anderen Zeichenkette
- Der String `suche` wird in dem String `sucheIn` gesucht.
- Falls die Suche erfolgreich war, wird ein Zeiger auf das erste Zeichen von `suche` in `sucheIn` zurückgeliefert. Andernfalls wird ein NULL-Zeiger zurückgeliefert.
- Es wird die Groß- und Kleinschreibung berücksichtigt.

# Beispiel

```
char puffer[8] = {"Eisbaer"};
char suche[8] = {"bae"};
char *ptr;

ptr = strstr(puffer, suche);

if ( NULL== ptr) {
     printf("%s wurde nicht gefunden", suche);
}
else {
     printf("Index: %d", (int) (ptr - puffer));
}
```

# Länge eines Teilstrings in Abhängigkeit eines Zeichens ermitteln



`size_t strcspn(const char *string, const char *liste);`

- Der String `string` wird zeichenweise eingelesen, bis eines der in `liste` enthaltenen Zeichen auftritt.
- Es wird die Anzahl der von `string` eingelesenen Zeichen zurückgeliefert.
- Groß- und Kleinschreibung wird berücksichtigt.

```
char zeichen[8] = {"Eisbaer"};
int pos = 0;

pos = strcspn(zeichen, "T");   /* Rückgabe 7*/
pos = strcspn(zeichen, "E");   /* Rückgabe 0*/
pos = strcspn(zeichen, "e");   /* Rückgabe 5*/
pos = strcspn(zeichen, "sba"); /* Rückgabe 2*/
pos = strcspn(zeichen, "sEa"); /* Rückgabe 0*/
```

# Bestimmte Zeichen aus einer Liste suchen

R|R|Z|N

```
char *strpbrk(const char *string, const char liste);
```

- Der String `string` wird zeichenweise eingelesen, bis eines der in `liste` enthaltenen Zeichen auftritt.
- Die Funktion liefert einen Zeiger auf die erste Fundstelle eines der in `liste` enthaltenen Zeichen. Andernfalls wird ein NULL-Zeiger zurückgeliefert.
- Bei der Suche wird die Groß- und Kleinschreibung wird berücksichtigt.

# Beispiel

```
char puffer[8] = {"Eisbaer"};
char suche[8] = {"sbe"};
char *ptr;

ptr = strpbrk(puffer, suche);

if ( NULL == ptr) {
     printf("%s wurde nicht gefunden", suche);
}
else {
     printf("Index: %d", (int) (ptr - puffer));
}
```

# Zeichen suchen, welches nicht vorkommt



```
size_t strspn(const char *suche, const char *sucheIn);
```

- Sind die Zeichen von `suche` in `sucheIn` enthalten?
- Beginnend mit dem ersten Zeichen von `suche`, überprüft die Funktion, ob die Zeichen von `suche` in `sucheIn` enthalten sind.
- Die Funktion stoppt, sobald ein Zeichen aus `suche` nicht in `sucheIn` enthalten ist.
- Die Funktion liefert die Anzahl der Zeichen, die von dem String `sucheIn` in `suche` enthalten sind. Es wird die Position in `suche` zurückgeliefert, welches keinem Zeichen aus `sucheIn` entspricht.

# Beispiel

```
char zeichen[5] = {"ABCE"};
char reihe[5] = {"ABFG"};
size_t pos;

pos = strspn(zeichen, reihe);

if (pos == 0) {
   printf("Keine Übereinstimmung");
}
else {
   printf("Übereinstimmung ab %d", pos);
}
```

# String anhand von Trennzeichen zerlegen



```
char *strtok(char *string, const char trennzeichen);
```

- Der String wird mit Hilfe von `trennzeichen` in Teilstrings zerlegt.
- `trennzeichen` können bei jedem Aufruf von `strtok()` beliebig verändert werden.
- Jedes vorhandene `trennzeichen` wird durch das String-Endezeichen ersetzt. Die Zeichenkette `string` wird bei jedem Aufruf verändert.
- In einer Schleife kann immer nur ein String in Teilstrings zerlegt werden.
- Zwischen den angegebenen Trennzeichen muss in `string` mindestens ein Zeichen stehen. Leere Strings werden nicht zurückgeliefert. Statt einem leeren String wird der nächste nicht leere Teilstring zurückgeliefert.

# Ablauf der Funktion

```
char *strtok(char *string, const char trennzeichen);
```

- Erster Aufruf der Funktion:
  - Beim ersten Aufruf der Funktion wird nach den angegebenen Trennzeichen gesucht.
  - Falls ein Trennzeichen gefunden wurde, wird ein Zeiger auf den ersten Teilstring zurückgeliefert.
  - Der gefundene Teilstring wird automatisch mit einem Endezeichen abgeschlossen
- Alle nachfolgenden Aufrufe der Funktion:
  - Die Funktion wird mit NULL statt der Zeichenkette `string` aufgerufen. Die Funktion hat sich den Inhalt von `string` gemerkt
  - Es werden alle nachfolgenden Teilstrings geliefert.

# Beispiel

R|R|Z|N

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

int main () {
  char nummer[] = "05022-45678\t06022-12345\n07045-45678\t456-789";
  char* result;

  result = strtok(nummer,"\n\t ");

  while( result  != NULL ) {
        printf("<%s>\n", result);
        result=strtok(NULL,"\n\t ");
  }

  return 0;
}
```

# Umwandlung in eine numerische Variable

R|R|Z|N

- Funktionen für die Umwandlung von Zeichen in Zahlen befinden sich in der Datei *stdlib.h*.
- Die Umwandlung wird so lange fortgeführt, bis ein nicht konvertierbares Zeichen gefunden wird.
- Funktionen:
  - Zeichen -> Int-Wert: `int atoi(char *ptr).`
  - Zeichen -> Long-Wert: `int atol(char *ptr).`
  - Zeichen -> Double-Wert: `int atof(char *ptr).`

# Umwandlung von Zeichen

R|R|Z|N

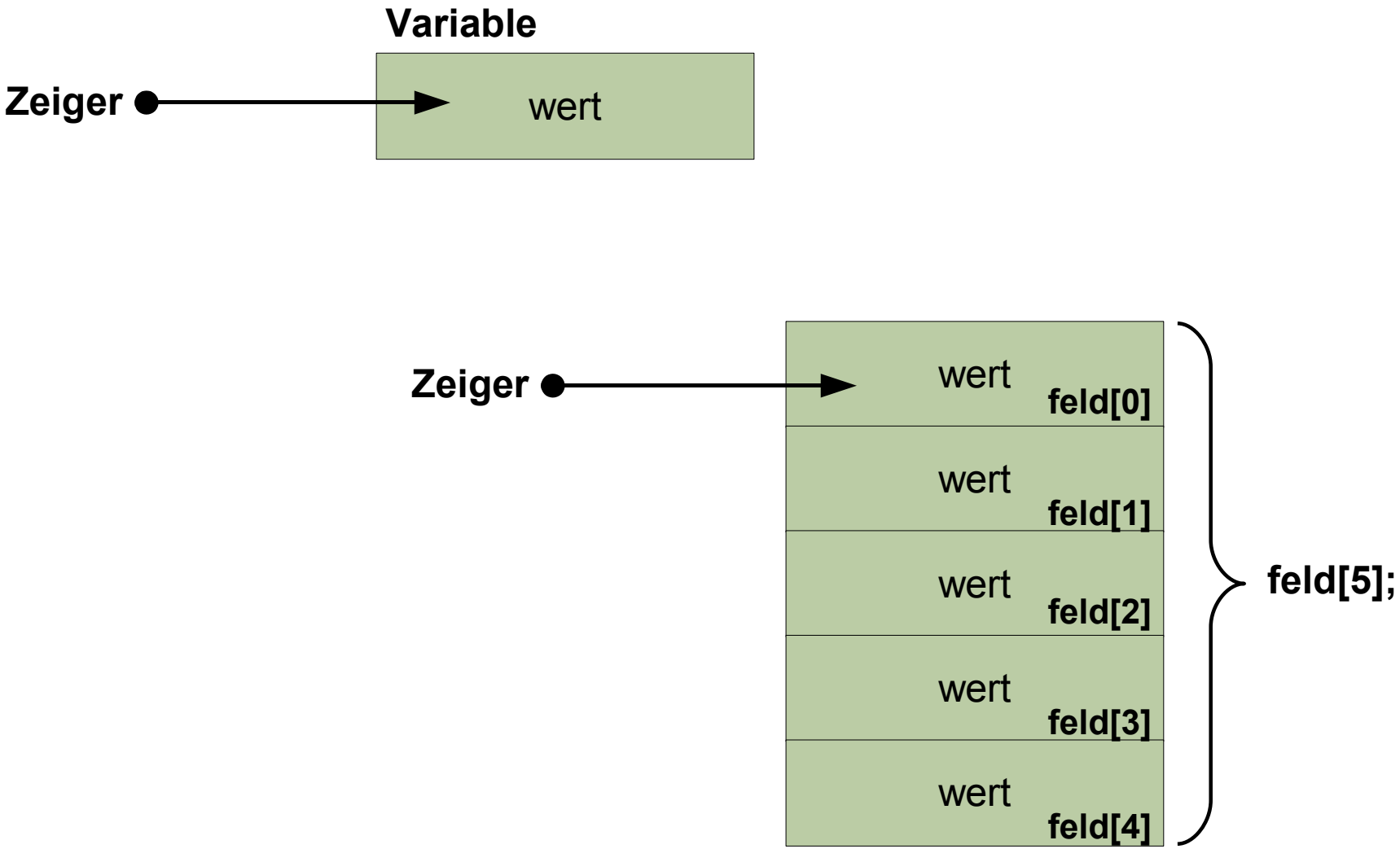
- Die Datei *ctype.h* enthält Funktionen zur Umwandlung von Groß- in Kleinbuchstaben und umgekehrt.
- Funktionen:
  - `int toupper(int chr)` Kleinbuchstaben -> Großbuchstaben.
  - `int tolower(int chr)` Großbuchstaben -> Kleinbuchstaben.
- Wenn die Schreibweise korrekt ist oder es sich um ein nicht alphabetisches Zeichen handelt, wird das Zeichen unverändert zurückgegeben.
- Das Zeichen wird als Integer der Funktion über- und  zurückgegeben.

# Eigenschaften von Zeichen testen



- Die Bibliothek *ctype.h* enthält Funktionen zum Testen von Zeichen.
- Alle Funktionen liefern einen boolschen Wert zurück.
- Funktionen

| | |
|---|---|
| `isalnum(char)` | Buchstabe oder Ziffer? |
| `isalpha(char)` | Buchstabe? |
| `islower(char)` | Kleinbuchstabe? |
| `isupper(char)` | Großbuchstabe? |
| `isdigit(char)` | Ziffer? |
| `isascii(char)` | Standard-ASCII-Zeichen (zwischen 0 und 127)? |
| `iscntrl(char)` | Steuerzeichen? |
| `isgraph(char)` | Druckbares Zeichen, aber kein Leerzeichen? |
| `isprint(char)` | Druckbares Zeichen einschließlich des Leerzeichens? |
| `ispunc(char)` | Leerzeichen? |
| `isspace(char)` | Leerzeichen, horizontaler oder vertikaler Tabulator, Zeilenvorschub, Seitenvorschub oder Wagenrücklauf? |
| isxdigit(char) | Hexadezimale Zahl? |


Variable
Zeiger
wert
Zeiger
wert
feld[0]
wert
feld[1]
wert
feld[2]
wert
feld[3]
wert
feld[4]
feld[5];

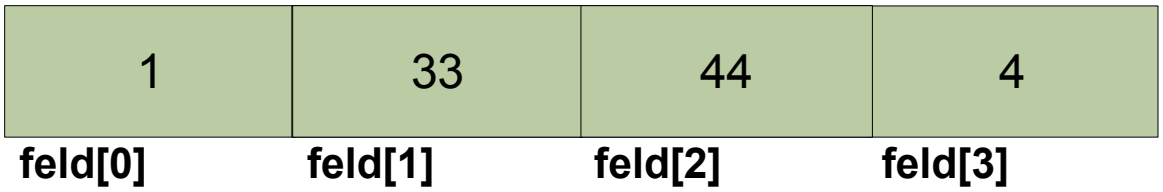
# Mit Zeigern auf Arrays zugreifen

R|R|Z|N

| | Array | Zeiger |
|---|---|---|
| Deklaration | Datentyp feldName[MaxAnzahl] | Datentyp *zeiger |
| Zugriff auf das erste Feldelement | | zeiger = feldName oder<br>zeiger = &feldName[0] |
| | feldName[0] = 4 | *zeiger = 4 |
| Zugriff auf das nächste Feldelement | feldName[1] = 5 | *(zeiger + 1) = 5 |

- Der Zeiger und das Objekt, worauf er verweist, sollten den gleichen Datentyp besitzen. Andernfalls können Fehler auftreten.
- Falls mit Zeigern gerechnet wird, wird immer der benötige Speicherplatz addiert oder subtrahiert. Wenn ein Zeiger, der auf ein Integer zeigt, addiert wird, wird der Zeiger immer um den Speicherplatz für ein Integer erhöht.
- Der Dereferenzierungsoperator hat eine höhere Bindung als das mathematische Zeichen für die Addition. Aus diesem Grund muss Erhöhung des Zeigers um 1 in Klammern gesetzt werden.

# Beispiel




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

```
int *ptr;
```

```
ptr = feld;
```

```
*(ptr + 1) = 33;
```

```
*(ptr + 2) = 44;
```

# Arrays als Funktionsargumente

R|R|Z|N

- Übergabe als Array:

```
Datentyp function(Datentyp array[ ])
```

- Übergabe mit Hilfe eines Zeigers:

```
Datentyp function(Datentyp *zeiger)
```

- Beide Möglichkeiten sind gleich.
  - Es wird nur die Adresse des Arrays übergeben.
  - Von dieser Anfangsadresse aus können alle Elemente des Arrays erreicht werden.
  - Die Anzahl der Elemente ist der aufgerufenen Funktion nicht automatisch bekannt.
- Die Funktion kann mit Hilfe von function(&array[0]); oder function(array); aufgerufen werden.

# Beispiel

```
#define MAX 7

int addition(int *element) {
   int count;
   int ergebnis;
   ergebnis = 0;

   for(count = 0; count < MAX; count++) {
        ergebnis = ergebnis + *(element+count);
   }

   return ergebnis;
}

int main() {
   int reihe[MAX] = {1, 2, 3, 4, 5, 6, 7};
   int *ptr;
   int ausgabe;
   ptr = reihe;
   ausgabe = addition(ptr);
}
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| E | I | S | B | Ä | R | \0 | | |
| B | R | A | U | N | B | Ä | R | \0 |
| K | O | A | L | A | B | Ä | R | \0 |
| G | R | I | Z | Z | L | Y | \0 | |

- Mehrere Strings werden in einer Tabelle abgespeichert.
- Deklaration einer Stringtabelle:

```
char *array[] ={"String1", "String2", ..., "StringN"}
char *baeren[] = {"EISBÄR","BRAUNBÄR","KOALABÄR","GRIZZLY"};
```

- Der Zugriff auf die einzelnen Elemente einer Stringtabelle erfolgt wie bei einem mehrdimensionalen Array:

```
array[Index] = "String";
*baeren[0] = "Eisbär";
```

# Stringtabellen



- Mit Hilfe des Index wird auf die Anfangsadressen der einzelnen Zeichenketten verwiesen.
- Mit Hilfe der Anweisung `array[index] + n` wird ein String ab einer bestimmten Position eingelesen.
- Zum Beispiel `baeren[0] + 3` liefert den String "bär" zurück.
  - `baeren[0]` verweist auf die Anfangsadresse des ersten Strings in der Tabelle.
  - + 3 verschiebt den Zeiger um drei Zeichen. Der Zeiger zeigt auf die Adresse des vierten Buchstaben des ersten Strings in der Tabelle.