

# Informatik II

- mehrdimensionale Felder
- Zeiger auf Zeiger

**Irene Rothe**

Zi. B 241

[irene.rothe@h-brs.de](mailto:irene.rothe@h-brs.de)

Instagram: irenerothesdesign



Hochschule  
Bonn-Rhein-Sieg

**Vorlesung\_H\_FelderStringsZeiger**

Irene Rothe, [irene.rothe@h-brs.de](mailto:irene.rothe@h-brs.de)

# Wie macht die Klingel beim Ingenieurbüro?

Dr ing Dr ing Dr ing



# Das Marie Kondo Prinzip

→ If it doesn't spark joy, get rid of it.

Wenn etwas keinen Spaß macht, ändere es!

If a function or line doesn't spark joy, get rid of it.

Quelle: <https://www.karim-geiger.de/blog/das-marie-kondo-software-design-principle>

# Informatik: 2 Semester für Ingenieure

## Informatik = Lösen von Problemen mit dem Rechner

✓ Zum Lösen von Problemen mit dem Rechner braucht man **Programmierfähigkeiten** (nur mit Übung möglich): Was ist Programmieren?

✓ Was ist ein Flussdiagramm?

### → **Programmiersprache C:**

✓ Elementare Datentypen

✓ Deklaration/Initialisierung

✓ Kontrollstrukturen: if/else, while, for

✓ Funktionen

✓ **Felder (Strings)**

→ **Zeiger**

→ struct

→ Speichieranforderung: malloc

→ Listen

→ Bitmanipulation

✓ Wie löst der Rechner unsere Probleme? → mit **Dualdarstellung** von Zeichen und Zahlen und mit Hilfe von **Algorithmen**



→ Ein Beispiel für ein Problem: **Kryptografie**

→ Sind Rechner auch Menschen? → **Künstliche Intelligenz**

→ Für alle Probleme gibt es viele Algorithmen. Welcher ist der Beste? → **Aufwand** von Algorithmen



# Design der Folien

- **Orange** hinterlegt sind alle Übungsaufgaben. Sie sind teilweise sehr schwer, bitte absolut nicht entmutigen lassen! Wir können diese in Präsenz besprechen oder über Fragen im Forum.
- **Grün** hinterlegte Informationen und grüne Smileys sind wichtig und klausurrelevant.
- Alles hinter „**Achtung**“ unbedingt beachten!
-  verwende ich, wenn überraschende Probleme auftreten können. Wenn Sie schon programmiererfahrend sind, können das eventuell besonders große Überraschungen für Sie sein, wenn Sie eine andere Sprache als C kennen.
- „Tipp“ benutze ich, um Ihnen einen Weg zu zeigen, wie ich damit umgehen würde.
- „Bemerkung“ in Folien beziehen sich meist auf Sonderfälle, die nicht unbedingt klausurrelevant sind, aber für Sie beim Programmieren eine Bedeutung haben könnten
-  hinter diesem Symbol ist ein Link fürs Anhören bzw. Gucken weiterer Infos

## Aufbau der Folien:

- Am Anfang motiviere ich gerne mit einem Beispiel, das eventuell schwer verständlich ist. Wem das nicht zusagt, dem empfehle ich, diese Folien zu überspringen.
- Weiter arbeite ich mit vielen Beispielen, die oftmals immer wieder das Gleiche erklären nur auf unterschiedliche Arten. Hat man einen Sachverhalt einmal verstanden, braucht man eventuell diese Beispiele nicht.
- Folien, die mit **Einschub** beginnen, beinhalten Zusatzinformationen, die nicht nötig für das Verständnis des Themas sind.
- Grün hinterlegte Informationen sind das, was Sie aus der Vorlesung rausnehmen sollen, alles andere sind vertiefende Informationen und Motivation.



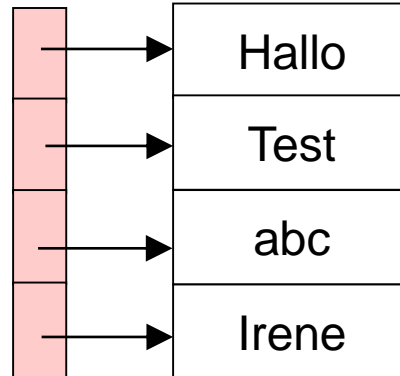
# Zeiger auf Zeiger (Felder): Motivation für Zeiger auf Felder - Wörter sortieren

Wie könnte ich praktischer Wörter sortieren, ohne alle Buchstaben der Wörter im Speicher zu bewegen?

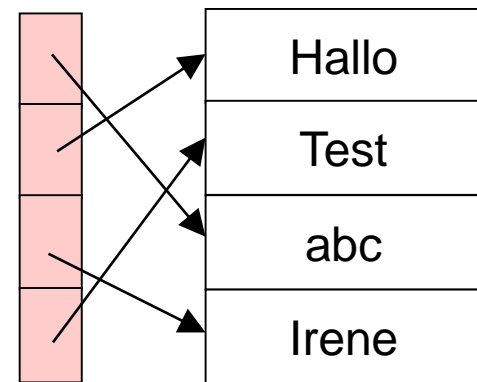
Ausgangssituation:

Hallo
Test
abc
Irene

Verwendung von Zeigern:



Sortierung:





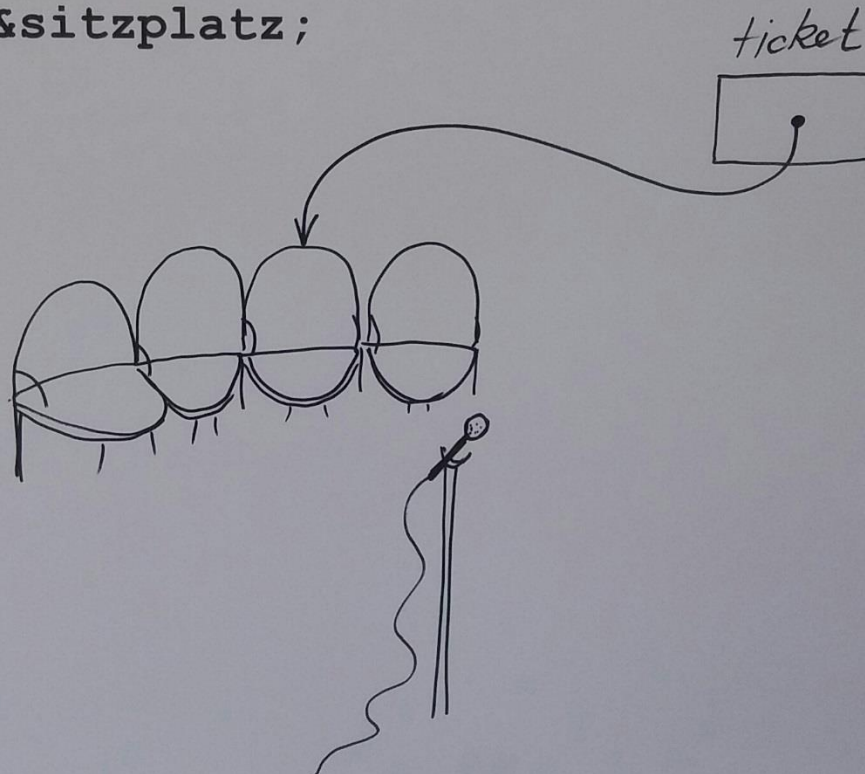
1. Wiederholung: Zeiger, Felder
2. Zusammenhang Felder und Zeiger
3. Zweidimensionale Felder
4. Funktionen und Zeiger
5. Funktionen mit Feldern
6. Strings
7. Felder von Zeigern





# Zeiger: Motivation - im Konzert

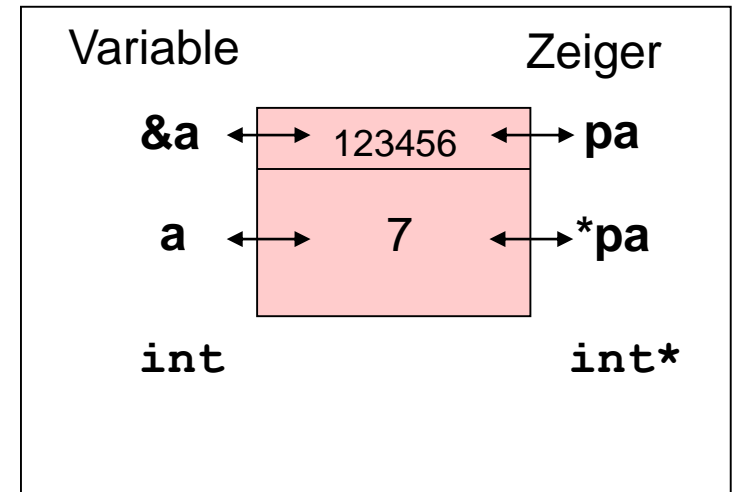
```
int sitzplatz;  
int *ticket;//sitzplatznummer  
ticket=&sitzplatz;
```



# 1. Wiederholung: Zeiger (Pointer)

- Adressen sind auch Werte, also kann man sie abspeichern.
- Ein Zeiger ist eine Variable, die eine Adresse beinhaltet.
- Ein Zeiger verweist damit auf den Speicherplatz (Adresse) einer anderen Variablen.
- Bei der Deklaration eines Zeigers muss feststehen, welcher Datentyp (int, char, double,...) auf der Adresse stehen darf.

	Int-Variable	Int-Zeiger
Deklaration	<code>int a;</code>	<code>int *pa;</code>
Nummer Speicherstelle	<code>&amp;a</code>	<code>pa</code>
Inhalt Speicherstelle	<code>a</code>	<code>*pa</code>



Man erkennt *deklarierte* Zeiger am Stern (\*) nach dem Datentyp.

**Achtung:** Zeigern muss eine gültige Adresse zugewiesen werden. Sonst besteht die Gefahr eines Computerabsturzes!

`pa=&a;` //Zuweisung einer Adresse an einen Zeiger durch Adressoperator



# 1. Wiederholung Zeiger: \* und & Operator

Mit dem \*-Operator kann man den Inhalt der Adresse, auf die ein Zeiger zeigt, abfragen. Diesen Operator nennt man *Dereferenzierungsoperator*.

Mit dem &-Operator kann man die Adresse einer Variablen abfragen.

\* und & heben sich auf: `*&wert=wert`

```
int main()
{
    int *zeiger;
    int a;
    zeiger=&a;
    a=6;
    *zeiger=7;
    printf("a = %i",a);    //Ausgabe: 7
    return 0;
}
```



[https://youtu.be/-2W\\_CU-0-rQ](https://youtu.be/-2W_CU-0-rQ)

zeiger

2

Adresse: 1

a

6

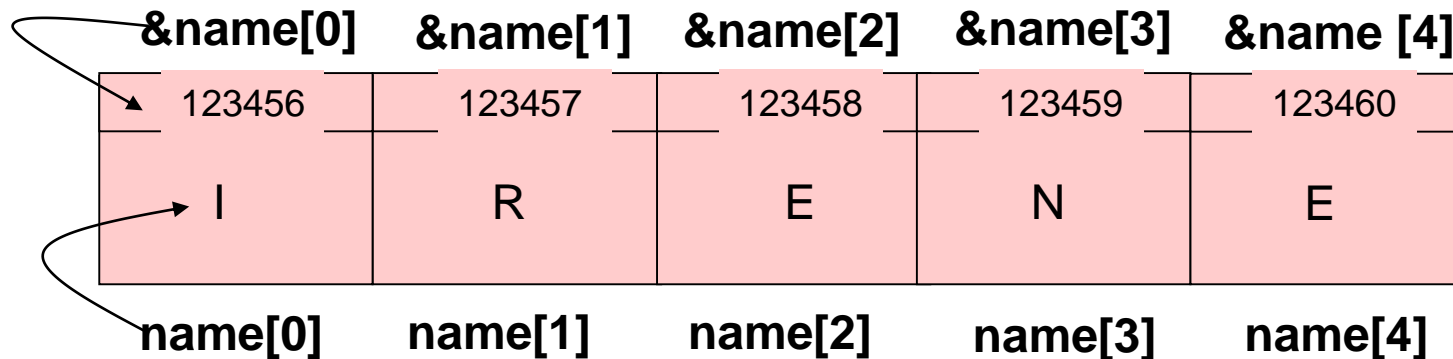
Adresse: 2

**Achtung:** Das sind Beispiele für Adressen, die der Rechner vergeben könnte.



# 1. Wiederholung Felder

- Wunsch: mehrere **gleiche** Dinge **hintereinander** abspeichern
- Zum Beispiel 5 Buchstaben: IRENE
- `char name[5];`



```
for (i=0; i<5; i++) {  
    printf ("%c", name[i]);  
}
```



<https://youtu.be/fBm7wyZ4mzs>

1. ✓ Wiederholung: Zeiger, Felder
2. Zusammenhang Felder und Zeiger
3. Zweidimensionale Felder
4. Funktionen und Zeiger
5. Funktionen mit Feldern
6. Strings
7. Felder von Zeigern



## 2. Zusammenhang Felder und Zeiger

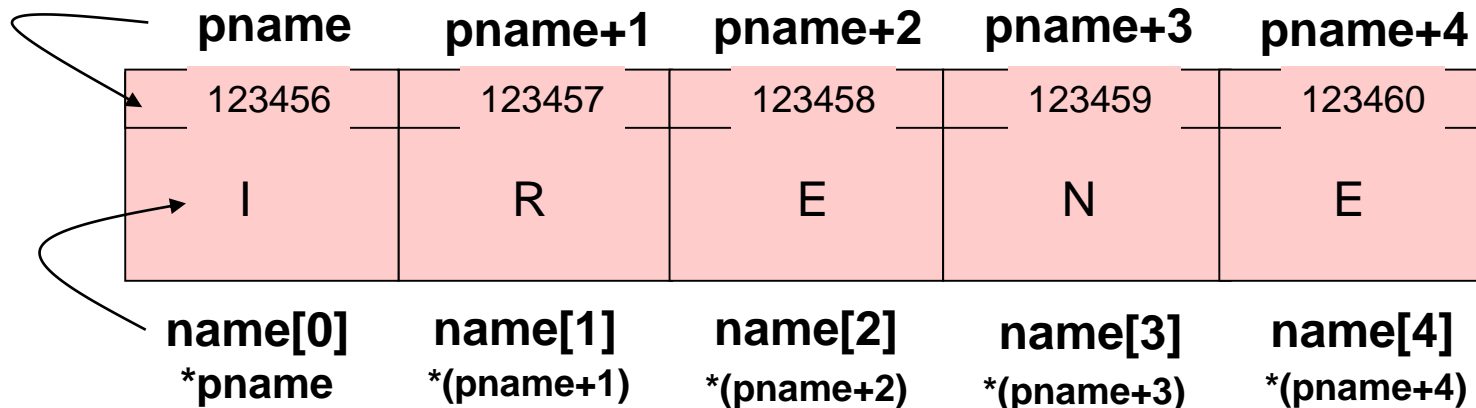
Felder sind auch mit Zeigern zu bearbeiten, da der Name eines Feldes die Adresse des ersten Elementes beinhaltet.

```
char name[5];  
char *pName;  
pName = &name[0];
```

O  
D  
E  
R

```
char name[5];  
char *pname;  
pname = name;
```

In C automatisch  
erzeugter konstanter  
Zeiger. Der wird immer erzeugt,  
wenn man ein Feld anlegt.  
Er kann nicht geändert werden.



## 2. Beispielprogramm

Verschiedene Varianten, wie man auf ein Feld zugreifen kann (1,2,4):

```
#include <stdio.h>
int main()
{
    char name[5]={'I','R','E','N','E'};
    //char *pname;
    //pname=name; //pname=&name[0]; geht auch
    int i;

    for(i=0;i<5;i++){
        //1: printf("%c",*(pname+i));
        //2: printf("%c",*pname);
        //2: pname++;
        //3: name++; geht nicht!
        //printf("%c",*(name+i)); //auch ueblich
        printf("%c",name[i]); //UEBLICH
    }
    return 0;
}
```



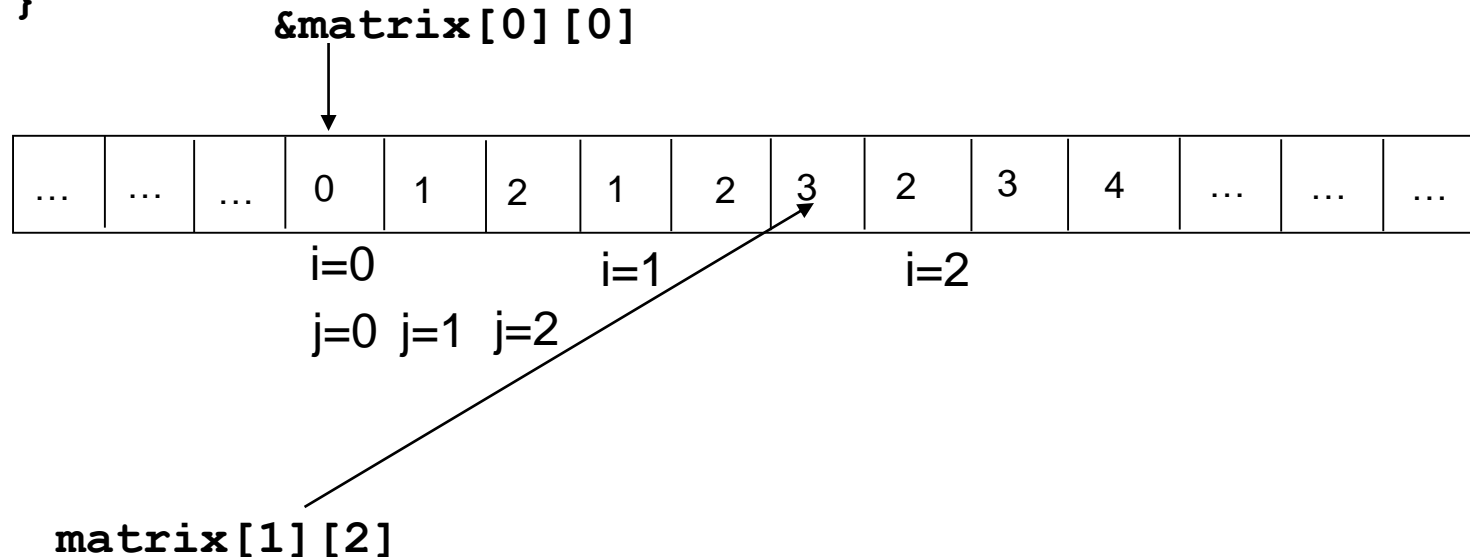


1. ✓ Wiederholung: Zeiger, Felder
2. ✓ Zusammenhang Felder und Zeiger
3. Zwei- und mehrdimensionale Felder
4. Funktionen und Zeiger
5. Funktionen mit Felder
6. Strings
7. Felder von Zeigern



# 3. Zweidimensionale Felder

```
int matrix[3][3]; //zweidimensionales Feld
for(i=0;i<3;i++){           //aussere Schleife
    for(j=0;j<3;j++){       //innere Schleife
        matrix[i][j]=i+j;
    }
}
```



### 3. Zweidimensionale Felder: aufwändiger

Vertauscht man in den Schleifen die Zeilen (i) mit den Spalten (j), hat der Rechner mehr zu tun, da Felder zeilenweise hintereinander abgespeichert sind. Der Rechner muss dann aufwändiger im Speicher hin- und herhüpfen.

```
int matrix[3][3];  
for(j=0;j<3;j++){  
    for(i=0;i<3;i++){  
        matrix[i][j]=i+j;  
    }  
}
```

...	...	...	0	1	2	1	2	3	2	3	4	...	...	...
-----	-----	-----	---	---	---	---	---	---	---	---	---	-----	-----	-----

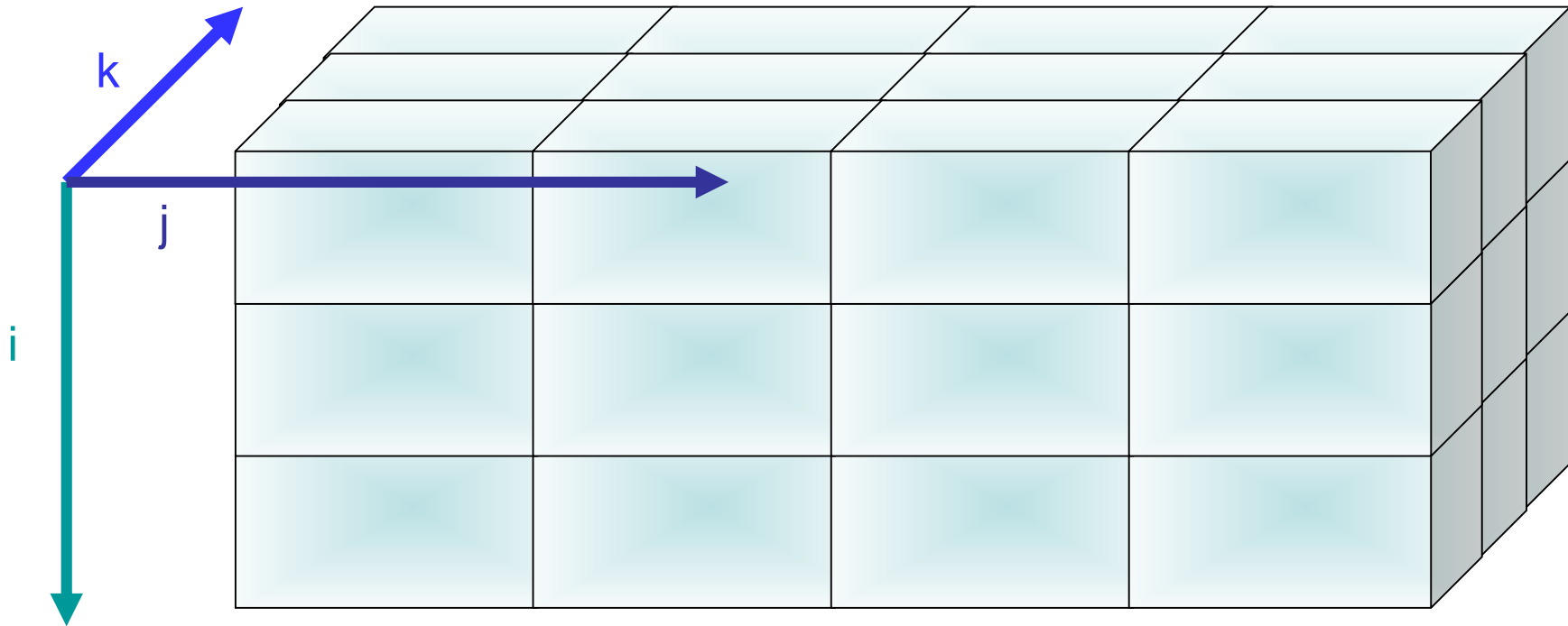
i=0

i=1

i=2

j=0 j=1 j=2

### 3. Dreidimensionales Feld



```
for (k=0;k<3;k++){  
    for (i=0;i<3;i++){  
        for (j=0;j<4;j++){  
            wuerfel[k][i][j]=i+j+k;  
        }  
    }  
}
```

# Zweidimensionale Felder: Übung

Es sei eine ganzzahlige 100 x 100 Matrix gegeben.

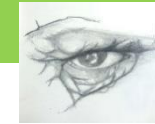
- a) Wie sieht die Deklaration dieser Matrix aus?
- b) Schreiben Sie C-Programmcode, der die erste Spalte der Matrix ausgibt!
- c) Schreiben Sie C-Programmcode, der die Diagonale der Matrix ausgibt!
- d) Schreiben Sie C-Programmcode, der die letzte Zeile der Matrix ausgibt!

Wenn Sie Fragen zur Lösung haben, schicken Sie mir Ihren Code!



# 3. Felder: Zusammenfassung

- Deklaration eines Feldes: Reservierung von hintereinander liegenden gleich großen Speicherplätzen
- Anzahl der Speicherplätze **muss** beim Übersetzen bekannt sein
- **Schrecklichkeit von C**: keine Feldgrößenüberprüfung: Sie sind verantwortlich, Ihr Feld nur soweit zu beschreiben, wie es Ihnen gehört.



- Mit Feldern arbeiten in der „Zeigerversion“:  $*(a+3)=42$ ;
- Mit Felder arbeiten in der „Indexversion“:  $a[3]=42$ ;

Ganz schlimm:

```
int feld[5];  
feld[5]=8; // ganz falsch!!!
```



1. ✓ Wiederholung: Zeiger, Felder
2. ✓ Zusammenhang Felder und Zeiger
3. ✓ Zweidimensionale Felder
4. Funktionen und Zeiger (Call by Reference)
5. Funktionen mit Felder
6. Strings
7. Felder von Zeigern





# 4. Funktionen: Wiederholung

- Funktionen sind praktisch, wenn Programmcode (Aufgabe) mehrmals verwendet wird.
- Funktionsname sollte Kurzfassung der Aufgabe sein.
- Aufgabe hat eventuell Eingabeparameter.
- Nur **ein** Wert kann zurückgegeben werden.



## 4. Funktionen: Beispiel - *Tausch* mit Zeigern

```
void tausch3(int *pa, int *pb){
    int help;
    help = *pa;
    *pa = *pb;
    *pb = help;
}

int main(){
    int a = 42, b = 17;
    printf("Davor: a=%i, b=%i\n",a,b);
    tausch3(&a,&b);
    printf("Danach: a=%i, b=%i\n",a,b);
    return 0;
}
```

**Achtung:** Mit einer return Anweisung in der Funktion **tausch** ist dieses Problem nicht zu lösen!



# 4. Funktionen: Call by Reference/ Call by Value

Eingabewerte können an Funktionen auf zwei Arten übergeben werden:

Call by Value: `double add1(double x, double y)`

Call by Reference: `double add2(double *x, double *y)`

```
int main() {  
    double a=2,b=4,c,d;  
    // ...  
    c = add1(a,b);  
    d = add2(&a,&b);  
    return 0;  
}
```

```
double add1(double x, double y) {  
    return x+y;  
}
```

```
double add2(double *x, double *y) {  
    return *x+*y;  
}
```

**Call by Value:** die aufgerufene Funktion arbeitet mit einer Kopie des Eingabewertes und kann ihn deshalb nicht ändern

**Call by Reference:** soll doch ein Eingabewert einer Funktion geändert werden, muss die **Adresse** der Variablen übergeben werden

# Call by Reference: Übung

- a) Implementieren Sie eine C-Funktion, die zwei Eingabewerte x und y erhält und die Summe, das Produkt und die Differenz von x und y „zurückgibt“!
- b) Wie würde die Funktion in einem Hauptprogramm aufgerufen werden?

Achtung: Auf der nächsten Folie steht die Lösung!




# Lösung:

Funktionskopf:

```
int operation(int x, int y, int *s, int *prod)
```

Aufruf:

```
int main() {  
    int a=2,b=4,summe,produkt,differenz;  
    // ...  
    differenz = operation(a,b,&summe,&produkt);  
    printf("%i,%i,%i",differenz,summe,produkt);  
    return 0;  
}
```



```
int operation(int x, int y, int *s, int *prod){  
    *s=x+y;  
    *prod=x*y;  
    return x-y;  
}
```

# Call by Reference: Übung - Wechselgeld

```
int rest=0; //Variable muss global sein, da sie von "main" und von "rueckgeld"
           //benutzt wird

int berechneRueckgeld(int wert){
    int i=0;
    while(rest>=wert){
        rest=rest-wert;
        i=i+1;
    }
    return i;
}

int main(){
    anzahl_01=0, anzahl_02=0, anzahl_50=0;
    printf("Rueckgeld in Cent: ");
    scanf("%i",&rest);
    anzahl_50 = berechneRueckgeld(50);
    anzahl_02 = berechneRueckgeld(2);
    anzahl_01 = berechneRueckgeld(1);
    printf("Rueckgabe:%i\n 50-Cent Stuecke: %i\n 2-Cent Stuecke: %i \n 1 Cent
           Stuecke: %i",anzahl_50, anzahl_02, anzahl_01);

    return 0;
}
```



# Call by Reference: Übung - Wechselgeld

Wie könnte man die Funktion **berechneRueckgeld** implementieren, ohne eine globale Variable zu benutzen?

Achtung: Auf der nächsten Folie steht die Lösung!





# Lösung: Wechselgeld

```
int berechneRueckgeld(int wert, int *rest){
    int i=0;
    while(*rest>=wert){
        *rest=*rest-wert;
        i=i+1;
    }
    return i;
}

int main(){
    int rest=0, anzahl_01=0, anzahl_02=0, anzahl_50=0;
    printf("Rueckgeld in Cent: ");
    scanf("%i",&rest);
    anzahl_50 = berechneRueckgeld(50, &rest);
    anzahl_02 = berechneRueckgeld(2, &rest);
    anzahl_01 = berechneRueckgeld(1, &rest);
    printf("Rueckgabe:%i\n 50-Cent Stuecke: %i\n 2-Cent Stuecke: %i \n 1 Cent
           Stuecke: %i",anzahl_50, anzahl_02, anzahl_01);
    return 0;
}
```



1. ✓ Wiederholung: Zeiger, Felder
2. ✓ Zusammenhang Felder und Zeiger
3. ✓ Zweidimensionale Felder
4. ✓ Funktionen und Zeiger
5. Funktionen mit Feldern
6. Strings
7. Felder von Zeigern



# 5. Funktionen und Felder: Übergabe

Felder werden in Funktionen immer als Call by Reference übergeben. Das heißt, man muss das Feld nicht zurückgeben (was in C gar nicht geht) und doch ist das Feld beschrieben in der Funktion, wo eingebenFeld aufgerufen wurde, weil im originalem Feld gearbeitet wird.

```
void eingebenFeld(double feld[10])
```

Aufruf der Funktion eingebenFeld:

```
int main() {
```

```
    double f[10];  
    eingebenFeld(f);
```

```
    return 0;
```

```
}
```

Sieht aus wie Call by Value,  
Ist aber Call by Referenz: es  
wird die Adresse des  
ersten Elements des Feldes  
Übergeben.  
Da heißt:  
Felder werden auch in  
der aufrufenden Funktion  
geändert!

Bemerkung: Es wäre der blanke Wahnsinn, wenn man z.B. große Felder kopieren würde bei einem Funktionsaufruf.

# 5. Funktion mit Feld – eindimensionales Feld



Das hier ist ein großes Problem!  
Die 4 im Funktionskopf beachtet der Compiler nicht. Wenn man in der Funktion aber das Feld weiter beschreibt, als es vorher definiert wurde, überschreibt man Speicher, der einem nicht gehört.

```
void ausgebenFeld(int feld[4]){  
    int i=0;  
    for (i=0; i<4; i++){  
        printf("feld[%i] = %i\n",i,feld[i]);  
    }  
}  
  
int main(){  
    int array[]={4,2,5,17};  
    ausgebenFeld(array);  
    return 0;  
}
```

Ein mögliche Lösung ist auf der nächsten Folie angegeben, aber wasserfest ist die nicht. Der Nutzer der Funktion wird aber so wenigstens aufmerksam gemacht, dass die Größe stimmen muss.



# 5. Funktion mit Feld – eindimensionales Feld


Ein mögliche Lösung des Problems der vorhergehenden Folie, die aber nicht wasserfest ist:

```
void ausgebenFeld(int feld[], int laenge) {  
    int i=0;  
    for (i=0; i<laenge; i++){  
        printf("feld[%i] = %i\n",i,feld[i]);  
    }  
}  
  
int main() {  
    int array[]={4,2,5,17};  
    ausgebenFeld(array,4);  
    return 0;  
}
```



## 5. Funktion mit Feld – mehrdimensionales Feld

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



Die Anzahl der Spalten muss hier angegeben werden. matrix ist ein Zeiger auf ein zwei-dimensionales Feld. Dies versteht der Compiler aber nur, wenn er weiß, wann eine neue Zeile beginnt.

# 5. Funktion mit Feld – mehrdimensionales Feld

Achtung: Das geht nicht



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



# Mehrdimensionale Felder: Übung

Bei der Definition einer Matrix muss die Anzahl der Spalten immer mit angegeben werden!

**Warum?**



# 5. Funktionen mit Zeiger als Rückgabewert

Achtung: Eine Funktion kann kein Feld zurückgeben!

Syntaktisch nicht möglich in C



Sehr gefährlich:

```
int * Gefaehrlich(...) {...} ;
```

Wenn eine Funktion eine Adresse zurückgibt, muss darauf geachtet werden, dass diese auch gültig ist!

**Tickende Zeitbombe!!!** Zeiger muss ja vorher lokal in der Funktion angelegt worden sein und mit einer Variable initialisiert. In dem Moment, wo sie zurückgegeben wird, ist die Adresse eigentlich schon wieder freigegeben.

Beispiel: 

```
int *Gefaehrlich() {  
    int lokalerMist=0;  
    return &lokalerMist;  
}
```



# Funktion und Felder: Beispiel 1 - größte ganze Zahl

```
#include <stdio.h>

int berechneGroessteZahl(int feld[], int anzahl){
    int gz,i;
    gz=feld[0];
    for(i=0;i<anzahl;i++){
        if(feld[i] > gz){
            gz=feld[i];
        }
    }
    return gz;
}

int main() {
    int zahlen[10];
    int ergebnis;
    int i, anzahl=10;
    printf("Bitte geben Sie nacheinander %i Zahlen ein:\n", anzahl);
    for (i=0;i<anzahl;i++){
        scanf("%i",&zahlen[i]);
    }
    ergebnis = berechneGroessteZahl(zahlen, 10); //call by reference
    printf("Die groesste Zahl ist %i\n", ergebnis);
    return 0;
}
```



# Funktion und Felder: Beispiel 2 - größte ganze Zahl

```
#include <stdio.h>

void berechneGroessteZahl(int feld[], int anzahl, int *gz){
    *gz=feld[0]; //Inhaltsoperator
    int i;
    for(i=1;i<anzahl;i++){
        if(feld[i] > *gz){
            *gz=feld[i];
        }
    }
}

int main() {
    int zahlen[10];
    int ergebnis;
    int i, anzahl=10;
    printf("Bitte geben Sie nacheinander %i Zahlen ein:\n", anzahl);
    for (i=0;i<anzahl;i++){
        scanf("%i",&zahlen[i]);
    }
    berechneGroessteZahl(zahlen, 10, &ergebnis); //call by reference
    printf("Die groesste Zahl ist %i\n", ergebnis);
    return 0;
}
```



# Bemerkung: main-Funktion mit Eingabe

Anzahl der Eingaben

Liste der Eingabewerte

```
int main(int argc, char *argv[]){  
    while (argc>0)    {  
        printf("%s \n",argv[argc-1]);  
        argc --;  
    }  
  
    return 0;  
}
```

In argv[0] steht der Programmname inklusive Pfad.



1. ✓ Wiederholung: Zeiger, Felder
2. ✓ Zusammenhang Felder und Zeiger
3. ✓ Zweidimensionale Felder
4. ✓ Funktionen und Zeiger
5. ✓ Funktionen mit Feldern
6. Strings
7. Felder von Zeigern

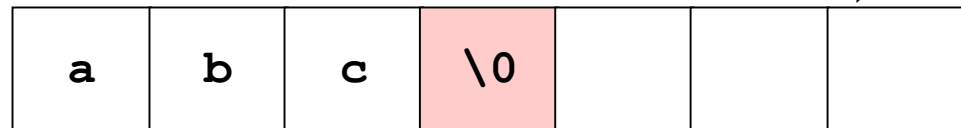
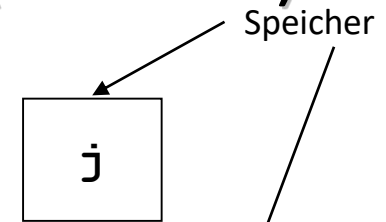


# 6. Strings (Zeichenketten) und Char (Zeichen)

Eine Zeichenkette ist ein Feld von char-Elementen.

Zeichen: `char a = 'j';`

Zeichenkette: `char line[100] = "abc";`



Nullzeichen

Ausgabe

eines Zeichens: `printf("%c \n", a);`

einer Zeichenkette: `printf("%s \n", line); // Ausgabe bis \0`

Der Formatierer s und das Endezeichen sind typisch für Strings.

## 6. Eingabe eines Strings

```
scanf("%s", line);
```

Hier wird Endezeichen automatisch mit angehängt.

**Achtung:** **line** muss groß genug vorher deklariert worden sein!





# 6. Bemerkung: Strings - Operationen der Bibliothek `string.h`

String kopieren von `s2` nach `s1`, Rückgabe Zeiger auf `s1`:

```
char *strcpy(char *s1, char *s2);
```

Kopie von String `s2` wird ans Ende von `s1` angehängt:

```
char *strcat(char *s1, char *s2);
```

Strings vergleichen:

```
int strcmp(char *s1, char *s2);
```

{	1	$s1 > s2$
	0	$s1 == s2$
	-1	$s1 < s2$

**Achtung:** Keine Prüfung der Größe!



Stringlänge:

```
unsigned int strlen(char *string);
```

## 6. Bemerkung: Strings – Beispiel mit `strcmp()`

```
#include <stdio.h>
#include <string.h>
int main(){

    int ergebnis;
    char eingabe[10];

    printf("Bitte geben Sie Ihren Namen ein:");

    scanf("%s", eingabe);

    if (strcmp(eingabe, "Irene") == 0){
        printf("Hallo Irene!");
    }
    else{
        printf("Hallo Fremder!");
    }
    return 0;
}
```

# 6. Bemerkung: Stringoperationen - Aufpassen

Vorsicht!

Stringoperationen prüfen **nicht** die Stringgröße



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

```
int main() {
    int i=0;
    char s1[]="abc";
    char s2[]="abcdefghi";
```

```
    strcpy(s1,s2); //kann/wird Probleme geben, da s2 größer s1
```

```
    printf("%s \n",s1);
    printf("%s \n",s2);
```

```
    printf("\ni = %i\n",i);
    return 0;
```

```
}
```



# 6. Strings: Zusammenfassung

```
char s1[] = "ABCDEF" ;
```

```
char s2[10] = "ABCDEF" ;
```

Reservierung von 7  
aufeinanderfolgenden char-  
Speicherplätzen, Zuweisung von:  
ABCDEF\0

Reservierung von 10  
aufeinanderfolgenden char-  
Speicherplätzen, Zuweisung von:  
ABCDEF\0

```
char *ps1 = s1;
```

Folgende Ausdrücke sind jeweils  
äquivalent:

<code>s1, ps1, &amp;s1[0]</code>
<code>s1+i, ps1+i, &amp;s1[i]</code>
<code>*(s1+i), *(ps1+i), s1[i]</code>

Achtung:

'x' – braucht 1 Byte

"x" – braucht 2 Byte

1. ✓ Wiederholung: Zeiger, Felder
2. ✓ Zusammenhang Felder und Zeiger
3. ✓ Zweidimensionale Felder
4. ✓ Funktionen und Zeiger
5. ✓ Funktionen mit Feldern
6. ✓ Strings
7. Felder von Zeigern

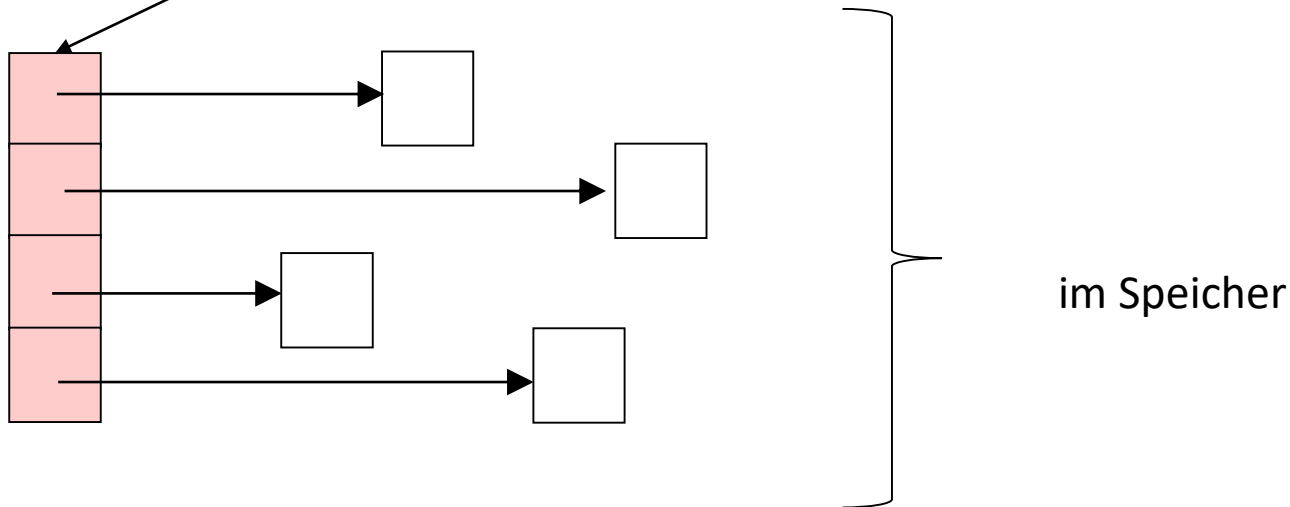


# 7. Felder von Zeigern

Felder sind Datenstrukturen, wo hintereinander gleiche Datentypen stehen. Also könnten in Feldern auch Adressen stehen, was bedeutet, dass man ein Feld von Zeigern deklarieren kann.

Deklaration: `typ *name[laenge];`

Beispiel: `int *c[4];` // c ist Zeiger aufs erste Element des Feldes



# 7. Felder von Zeigern: Beispiel

```
1: int a[] = {1,2,3};
```

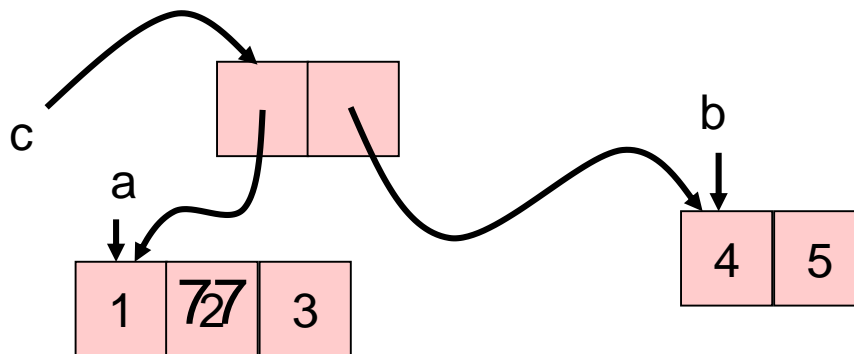
```
2: int b[] = {4,5};
```

```
3: int *c[2];
```

```
4: c[0]=a;
```

```
5: c[1]=&b[0];
```

```
6: *(c[0]+1)=77;
```



# 7. Felder von Zeigern: größeres Beispiel

Code:

```
int *hitzefeld[4];
int sonne;
int gluehtage[]={1,7,23};
int temperaturen[]={38,39,40,41};
char motto[]={ 'O', 'H', 'N', 'M', 'A', 'E', 'C', 'H', 'T', 'I', 'G' };
int bierverbrauch[]={25,30,40,100};

sonne = 120;
hitzefeld[2]= &sonne;
*hitzefeld = gluehtage;
hitzefeld[3]= temperaturen;
*(hitzefeld+1)=&bierverbrauch[0];

printf ("%c\n",*(motto+2));
printf ("%i\n",*(hitzefeld+1));
printf ("%i\n",**hitzefeld);
printf ("%i\n",*(hitzefeld[3]+1));
printf ("%c\n", motto[1]);
printf ("%i\n", hitzefeld[2][1]);
printf ("%i\n",*(*(hitzefeld+1)+1));
```

Blick in den Speicher:



<https://www.youtube.com/watch?v=J7z4D1K3z5M&t=20s>



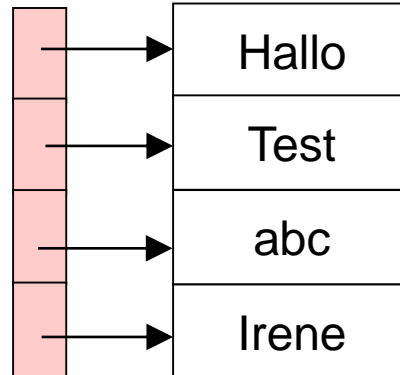
# 7. Felder von Zeigern: Wofür?

Zum Beispiel fürs Wörter sortieren ohne die Wörter selbst im Speicher zu bewegen

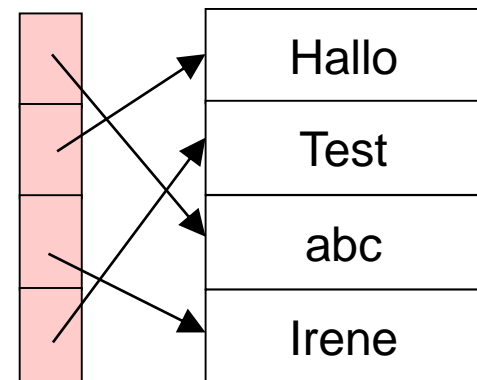
Ausgangssituation:

Hallo
Test
abc
Irene

Verwendung von Zeigern:



Sortierung:



In Präsenz: siehe ZeigerSortierBeispiel.c



# Sortierbeispiel: Bubblesort aus WS

```
void bubblesort (int f[], int anz) {
    int i,j,hilfe,sortiert=0;
    for ( i=0 ; i<anz; i=i+1 ) {
        if ( sortiert == 0 ) {
            // Annahme, dass Folge sortiert ist
            sortiert = 1;
            for ( j=0 ; j<anz-1 ; j=j+1 ){
                if ( f[j] > f[j+1] ) {
                    // Vertauschung notwendig
                    hilfe = f[j];
                    f[j]= f[j+1];
                    f[j+1]= hilfe;
                    // Folge war doch nicht sortiert
                    sortiert=0;
                }
            }
        }
    }
}
```

# Hauptprogramm von damals (WS)

```
int main(){
    int folge[6] = {4,6,5,1,2,9};
    int anzahl = 6;
    int i;
    bubblesort(folge, anzahl);
    for (i=0; i<anzahl; i=i+1) {
        printf("Folge: %i\n",folge[i]);
    }
    return 0;
}
```

# Sortierbeispiel: Hauptprogramm

```
#include<stdio.h>

int main() {
    int i;
    int anz=4;
    char wort1[]="Irene";
    char wort2[]="Hallo";
    char wort3[]="Test";
    char wort4[]="abc";
    //Zeigerfeld
    char *sortierfeld[4];
    //Zuweisung der Wörter aufs Zeigerfeld
    sortierfeld[0]=wort1;
    sortierfeld[1]=wort2;
    sortierfeld[2]=wort3;
    sortierfeld[3]=wort4;

    bubblesort(sortierfeld,4);

    return 0;
}
```



# Sortierbeispiel: veränderter Bubblesort

```
void bubblesort (char *f[], int anz) {
    int i,j,sortiert=0;
    char *hilfe;//muss jetzt eine Adresse sein, da Adressen getauscht werden sollen
    for (i=0;i<anz;i=i+1) {
        if (sortiert == 0) {
            //Annahme, dass Folge sortiert ist
            sortiert = 1;
            for (j=0;j<anz-1;j=j+1){
                //Test, ob getauscht werden muss abhängig vom Inhalt der Adresse
                if (*f[j] > *f[j+1]) {
                    //Vertauschung notwendig
                    //hier werden die Adressen getauscht, wenn der Inhalt der Adressen das
                    //alphabetisch erfordert
                    hilfe = f[j];
                    f[j]= f[j+1];
                    f[j+1]= hilfe;
                    //Folge war doch nicht sortiert, sortiert zurücksetzen auf 0
                    sortiert=0;
                }
            }
        }
    }
}
```

In Präsenz: Siehe Bubblesortstring1,2,3.c



# Sortierbeispiel: veränderter Bubblesort

```
void bubblesort (char *f[], int anz) {
    int i,j,sortiert=0;
    for (i=0;i<anz;i=i+1) {
        if (sortiert == 0) {
            // Annahme, dass Folge sortiert ist
            sortiert = 1;
            for (j=0;j<anz-1;j=j+1){
                //Test, ob getauscht werden muss abhängig vom Inhalt der Adresse
                if (*f[j] > *f[j+1]) {
                    // Vertauschung notwendig
                    //hier werden die Adressen getauscht, wenn der Inhalt der Adressen das
                    //alphabetisch erfordert
                    tausch3(&f[j],&f[j+1]);
                    //Folge war doch nicht sortiert
                    sortiert=0;
                }
            }
        }
    }
}

void tausch3(char **pa, char **pb){
    char *help; //es werden Adressen getauscht
    help = *pa;
    *pa = *pb;
    *pb = help;
}
```

Siehe Bubblesortstring1,2,3.c



# Zeiger: Übung

```
1.  int *pt1, *pt2;
2.  int var1 = 10;
3.  int var2 = 20;
4.
5.  pt1  = &var1;
6.  pt2  = pt1;
7.  *pt1 = *pt1 + 1;
8.  pt1  = &var2;
9.  (*pt1)++;
10. *pt2 = 15;
11. pt1 = &var1;
12. pt2 = &var2;
13. *pt2 = *pt1;
14. *pt2 = 30;
15. pt2 = pt1;
16. *pt2 = 30;
```

Frage: Sind **var1** und **var2** gleich oder nicht?

Bemerkung: Zur Überprüfung Ihrer Vermutung, führen Sie den Code einfach am Rechner aus.



# Zeiger auf Zeiger: Übung

```
1: int *gruselfeld[4];
2: int zahl;
3: int zisch[]={3,2,1};
4: int zapp[]={9,8};
5: char rambaZamba[]={'+','$', 'M'};
6: int vektor[]={27,18,0,42};
7: zahl=101010;
8: gruselfeld[2]=&zahl;
9: *gruselfeld =zisch;
10: gruselfeld[3]=vektor;
11: *(gruselfeld+1)=&zapp[0];
12: printf ("%c\n",*(rambaZamba+2));
13: printf ("%i\n",*(gruselfeld+1));
14: printf ("%i\n",**gruselfeld);
15: printf ("%i\n",*(gruselfeld[3]+2));
16: printf ("%c\n",rambaZamba[1]);
17: printf ("%i\n",*(*(gruselfeld +1)+1));
18: printf ("%i\n", gruselfeld [2][3]);
```

Bemerkung: Zur Überprüfung Ihrer Vermutung, führen Sie den Code einfach am Rechner aus.

Lösung eines ähnlichen Beispiels: [www.youtube.com/watch?v=J7z4D1K3z5M&t=20s](http://www.youtube.com/watch?v=J7z4D1K3z5M&t=20s)





# Zeiger: Übung

Was wird hier ausgegeben?

```
int p[2];
int *z;
z=&p[0];
p[0]=1;
p[1]=42;
printf("p[0]=%i p[1]=%i\n",p[0],p[1]);
(*z)++;
printf("p[0]=%i p[1]=%i\n",p[0],p[1]);
z++; //eine Adresse weiter
(*z)++;
printf("p[0]=%i p[1]=%i\n",p[0],p[1]);
```

Bemerkung: Zur Überprüfung Ihrer Vermutung, führen Sie den Code einfach am Rechner aus.



# Zeiger auf Zeiger speziell Strings: Übung

Was wird hier ausgegeben?

```
#include <stdio.h>

int main() {
    char *farbe[]={"WEISS", "PINK", "BLAU", "GRUEN"};

    1:  printf("%c ", *farbe[1]);
    2:  printf("%s ", *farbe);
    3:  printf("%c ", *(farbe[3]+2));
    4:  printf("%s ", farbe[2]+1);
    5:  printf("%c ", *(* (farbe+1)+3));
    return 0;
}
```

Bemerkung: Zur Überprüfung Ihrer Vermutung, führen Sie den Code einfach am Rechner aus.

# Zeiger auf Zeiger speziell Strings: Übung

Was wird hier ausgegeben?

```
#include <stdio.h>

int main() {

    char *dinge[]={ "PASS", "KUCHEN", "BAUM", "FUELLER", "GEIGE", "ZAHN", "EI" };

    printf("%c",*(dinge[2]+1));
    printf("%c",*(dinge[1]+1));
    printf("%c",** (dinge+3));
    printf("%s",*dinge);
    printf("%s ",*(dinge+1)+4);
    printf("%c",** (dinge+2));
    printf("%s ",*(dinge+6));
    printf("%c",dinge[5][0]);
    printf("%s",*(dinge+4)+1);
    printf("%c",* (* (dinge+3)+6));
    printf("%c",dinge[5][3]);
    return 0;
}
```

Bemerkung: Zur Überprüfung Ihrer Vermutung, führen Sie den Code einfach am Rechner aus.



# Funktionen: Zusammenfassung

Funktionen können wie folgt definiert werden (nicht vollständig, Beispiele):

- ohne Eingabegrößen und ohne Rückgabe:

```
void funktion() {...}
```

- mit Eingabegröße (call by value) und ohne Rückgabe:

```
void funktion (int eingabe){...}
```

- ohne Eingabegrößen und mit Rückgabe:

```
int funktion(){...}
```

- mit Feld als Eingabegröße (call by reference) und ohne Rückgabe:

```
void funktion(int feld[5]){...}
```

- mit Zeiger als Eingabegröße (call by reference) und mit Rückgabe:

```
int funktion (int *p){...}
```

- mit Zeiger als Eingabegröße (call by reference) und mit Zeiger als Rückgabe:

```
int * funktion(int *p){...}
```



# Literatur

- D. Bär: „Schrödinger programmiert C++“, Galileo Computing, 2012
- Jürgen Wolf: C von A bis Z - Das umfassende Handbuch, ISBN: 978-3-8362-1411-7

