

Dirac sagte: Die Wissenschaftler bemühen sich, noch nie Gesagtes so klar und einfach wie möglich zu formulieren -- die Lyriker machen genau das Gegenteil.



Informatik1

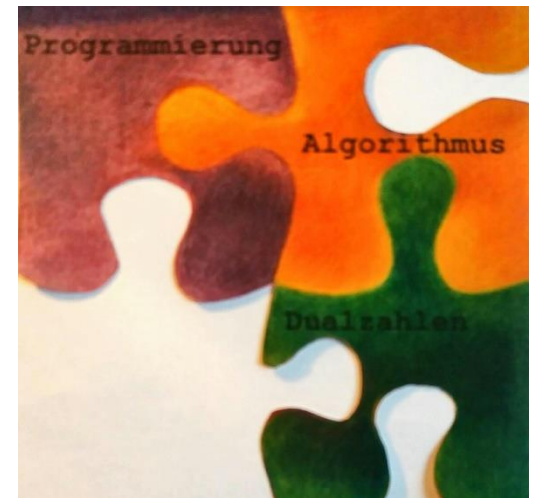
Felder, Zeiger (Funktionen mit Zeigern)

Irene Rothe

Zi. B 241

irene.rothe@h-brs.de

Instagram: irenerothedesign



Hochschule
Bonn-Rhein-Sieg

Vorlesung_E_ZeigerFelder

Abitur: Du bist allwissend

Bachelor: in einigen Bereichen hast Du
gute Grundkenntnisse

Master: Du hast ein sicheres Auftreten bei
völliger Unwissenheit

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



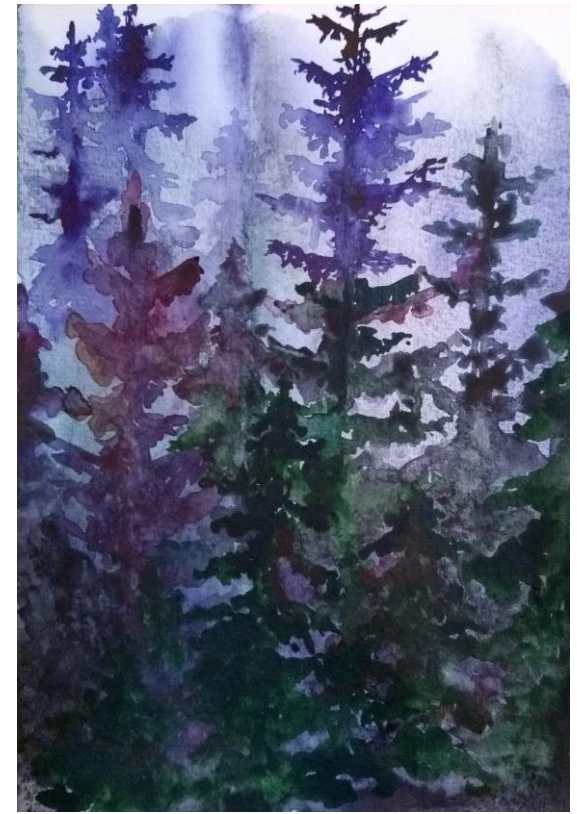
Design der Folien

-  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.
-  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

In dieser Vorlesung:

→ Felder

- eindimensionale Felder (Arrays)
- mehrdimensionale Felder
- Deklaration und Initialisierung von Feldern
- Zuweisung auf Felder



Zeiger (Funktionen mit Zeigern)

Felder: Motivation

Wir wollen Zahlen sortieren.

→ dafür brauchen wir nicht nur gute **Kontrollstrukturen**, sondern auch gute **Datenstrukturen**.

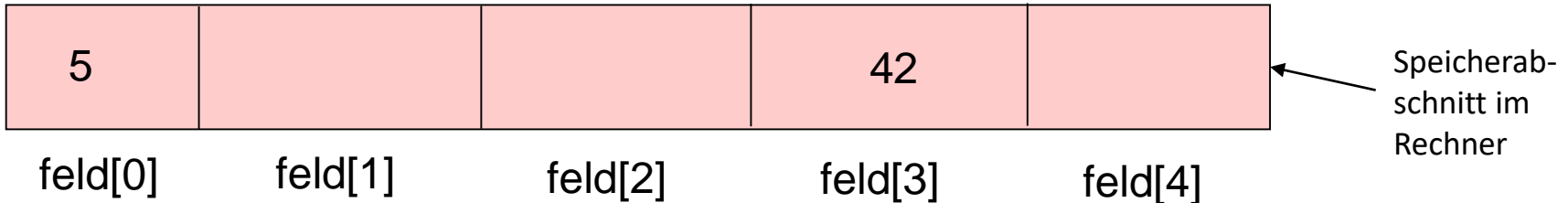


→ Eine Datenstruktur, wo man hintereinander Daten gleich Typs speichern kann, wäre gut.

Eindimensionale Felder (Arrays)

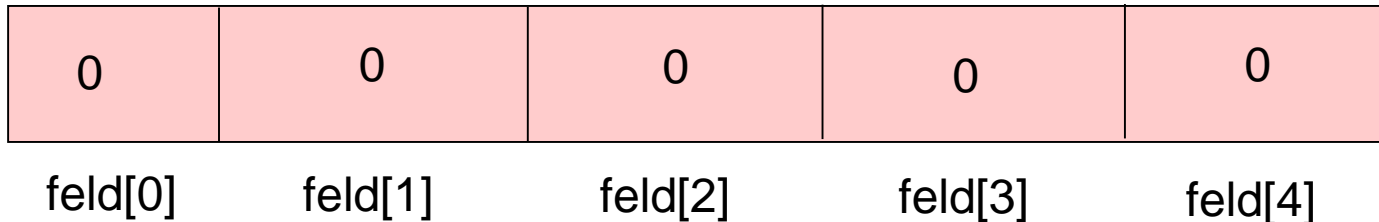
Felder fordern Speicherplatz für Daten vom **gleichen** Datentyp **hintereinander** an. Sie werden durch Indizes in eckigen Klammern *beginnend bei Null* angesprochen.

```
int feld[5]; //Deklaration
```



```
feld[0]=5;  
feld[3]=42;
```

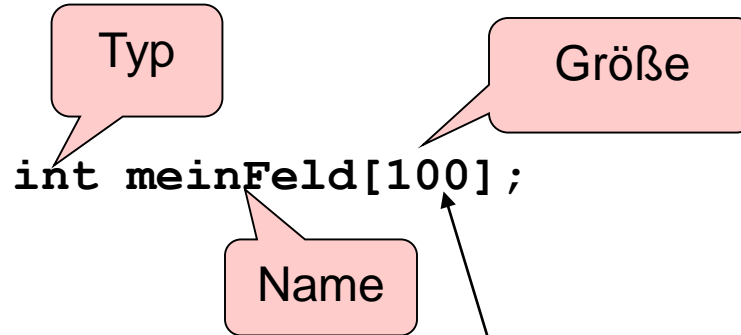
```
int feld[5]={0}; //Deklaration und Initialisierung
```



Achtung: **Felder haben eine konstante Größe in ANSI C!** 💣

Felder: Deklaration

Felder haben einen Datentyp, Namen und eine **fest** vorgegebene Größe.



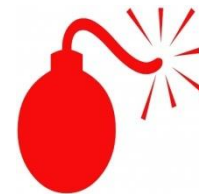
Achtung: Felder werden von **0** bis **Feldgröße-1** durchgezählt:

`meinFeld[0], meinFeld[1], ... meinFeld[99]`

Wertzuweisung und Ausgabe:

```
meinFeld[5]=17;  
printf("Inhalt %i",meinFeld[5]);
```

Achtung: Die Größe muss vorm Übersetzen festgelegt werden!



Felder: Initialisierung

Felder sind Variablen und werden wie üblich am Anfang deklariert und wenn gewünscht, initialisiert.

```
int feld[5]={0}; //geht nur für Null
int zahlen[] = {9,14,2,42,17}; //geht nur bei
                                //gleichzeitiger Deklaration
```

Oder so:

```
int zahlen[5]={0};
zahlen[0] = 9;
zahlen[1] = 14;
zahlen[2] = 2;
zahlen[3] = 42;
zahlen[4] = 17;
```

```
int zahlen[]; //Fehler
int zahlen[n]; //schlimmer Fehler in Ansi C
```



Felder: Wofür sind sie gut?

Man kann sehr praktisch Anweisungen über ein Feld laufen lassen in einer for-Schleife (man weiß immer wie groß ein Feld ist durch die Deklaration, dafür ist die for-Schleife perfekt).

Hier die Ausgabe eines Feldes **zahlen**:

```
int zahlen[5]={0}; //alle Speicherplätze werden mit 0 belegt
for (i=0;i<5;i++){
    printf("%i", zahlen[i]);
}
```

Ausgabe: 0 0 0 0 0



Mehrdimensionale Felder

Felder können mehrdimensional sein bis 32 (wofür auch immer das sinnvoll ist).

Beispiel 2-dimensional:

```
int matrix[5][3];  
matrix[0][0]=9;  
matrix[1][1]=5;  
matrix[4][2]=2;
```

(0,0)
→
↓

9		
	5	
		2

Felder: Beispiel - größte ganze Zahl

```
#include <stdio.h>

int main() {
    int zahlen[10];
    int groesstezahl;
    int i;
    printf("Bitte geben Sie nacheinander Zahlen ein:\n");
    for (i=0;i<10;i++){
        scanf("%i",&zahlen[i]);
    }
    groesstezahl=zahlen[0];
    for(i=1;i<10;i++){
        if(zahlen[i] > groesstezahl){
            groesstezahl=zahlen[i];
        }
    }
    printf("Die groesste Zahl ist %i\n", groesstezahl);
    return 0;
}
```



Felder: Tipp: Flexiblere Programmierung

```
#include <stdio.h>

int main() {
    int feldgroesse=10;
    int zahlen[feldgroesse]; //beim Compilieren steht hier
    int groesstezahl;          // wirklich eine Zahl!!
    int i;
    printf("Bitte geben Sie nacheinander Zahlen ein:\n");
    for (i=0;i< feldgroesse;i++){
        scanf("%i",&zahlen[i]);
    }
    groesstezahl=zahlen[0];
    for(i=1;i< feldgroesse;i++){
        if(zahlen[i] > groesstezahl){
            groesstezahl=zahlen[i];
        }
    }
    printf("Die groesste Zahl ist %i\n", groesstezahl);
    return 0;
}
```

Felder: Übung

Frage: Was wird hier ausgegeben? Ihre Vermutung können Sie dann überprüfen, in dem Sie das Programm ausführen lassen.

```
#include<stdio.h>
int main(){
    int zahlenfeld[3]={1,2,3};
    char zeichenfeld[4]={'A','B','C','D'};
    printf("zahlenfeld[0] = %i\n",zahlenfeld[0]);
    printf("zahlenfeld[2] = %i\n",zahlenfeld[2]);
    zahlenfeld[1]=88;
    printf("zahlenfeld[1] = %i\n",zahlenfeld[1]);
    printf("zeichenfeld[1] = %c\n",zeichenfeld[1]);
    printf("zeichenfeld[3] = %c\n",zeichenfeld[3]);
    printf("zeichenfeld[8] = %c\n",zeichenfeld[8]);

    return 0;
}
```



Felder: Übung

Geben Sie folgende Felder aus:

```
int main() {  
    int zahlenfeld[3]={1,2,3};  
    char zeichenfeld[4]='A','B','C','D';  
  
    return 0;  
}
```



Felder...

- beginnen immer bei Null
- in Schleifen am Besten immer wie folgt: `i=0 ; i < Feldgroesse ; i++`
- immer Angst haben vor Buffer-Overflow (sie schreiben auf Speicher, den Sie nie reserviert haben)!!!!

Was kann passieren bei Buffer-Overflow?

- Betriebssystem beendet das Programm wegen einer Speicherverletzung
→ (segmentation fault)
- Ist nicht ganz klar
- Nichts
- Irgendwann ein Absturz
- Weiterarbeit mit falschen Werten
- Speicherbeschädigung



✓ Felder

→ Zeiger

- Definition von Zeigern (Pointer)
- Verwendung im Zusammenhang mit Funktionen (Call by Reference)
- Gefahrenquellen bei Zeigern

Aufbau der Folien:

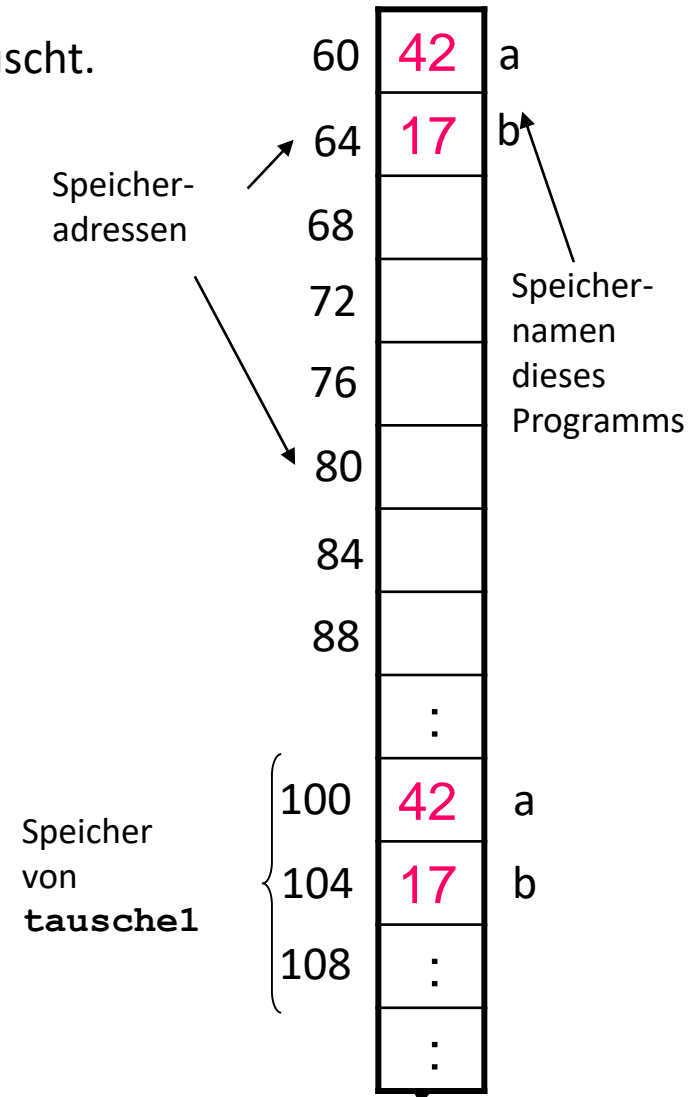
- Am Anfang motiviere ich gerne mit einem Beispiel, das eventuell schwer verständlich ist. Wem das nicht zusagt, dem empfehle ich, diese Folien zu überspringen.
- Weiter arbeite ich mit vielen Beispielen, die oftmals immer wieder das Gleiche erklären nur auf unterschiedliche Arten. Hat man einen Sachverhalt 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: Motivation - Funktion **tauschel**

Wunsch: Implementierung einer Funktion, die 2 Werte tauscht.

```
void tauschel(int a, int b){
    int help;
    help = a;
    a = b;
    b = help;
    printf("Tausch: a=%i, b=%i\n",a,b);
}

int main() {
    int a = 42, b = 17;
    printf("a=%i, b=%i\n",a,b);
    tauschel(a,b);
    printf("a=%i, b=%i\n",a,b);
    return 0;
}
```



Erster Versuch einer Tauschfunktion.

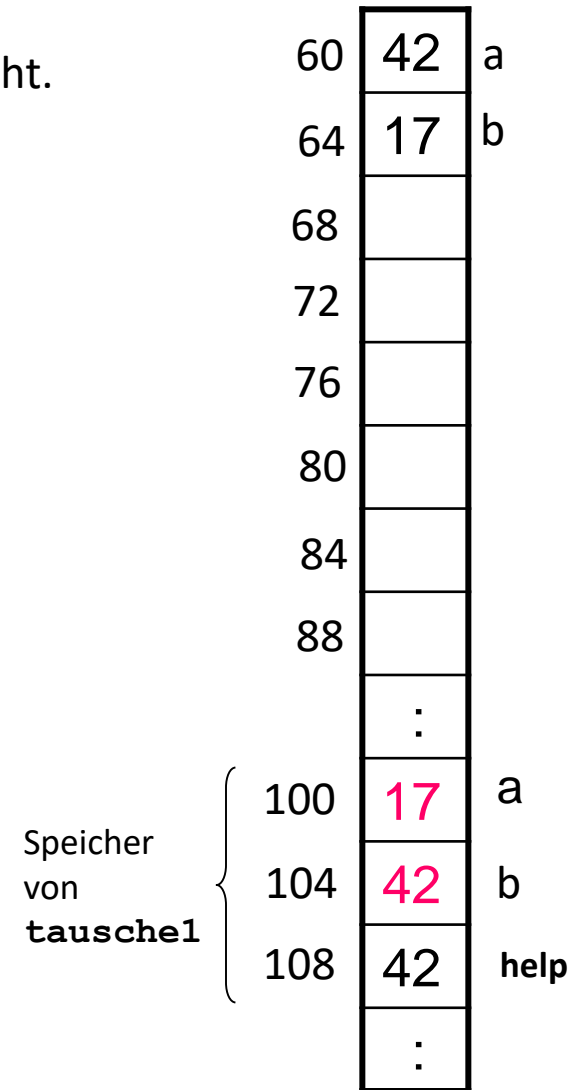


Zeiger: Motivation - Funktion `tauschel`

Wunsch: Implementierung einer Funktion, die 2 Werte tauscht.

```
void tauschel(int a, int b){
    int help;
    help = a;
    a = b;
    b = help;
    printf("Tausch: a=%i, b=%i\n",a,b);
}

int main() {
    int a = 42, b = 17;
    printf("a=%i, b=%i\n",a,b);
    tauschel(a,b);
    printf("a=%i, b=%i\n",a,b);
    return 0;
}
```



→ Erkenntnis: Diese Funktion macht nicht, was wir wollen!



Zeiger: Motivation - Funktion tausche2 (globale Variablen)

```
int a,b; //globale Variablen
void tausche2(){
    int help;
    help = a;
    a  = b;
    b  = help;
    printf("Tausch: a=%i, b=%i\n",a,b);
}
int main() {
    int a = 42, b = 17;
    printf("a=%i, b=%i\n",a,b);
    tausche2();
    printf("a=%i, b=%i\n",a,b);
    return 0;}
```

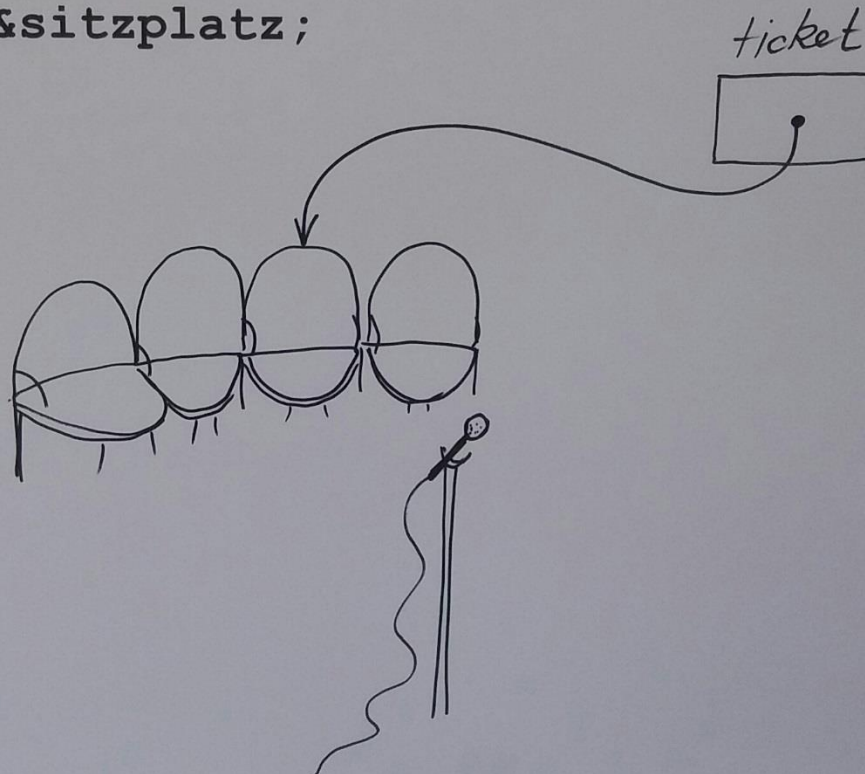
Besser mit Zeigern (Pointern)

Globale Variablen sind anstrengend! Ein Programm mit globalen Variablen ist sehr schwer zu überschauen. Man braucht keine globalen Variablen, deshalb kann man sie gleich wieder vergessen!



Zeiger: Motivation - im Konzert

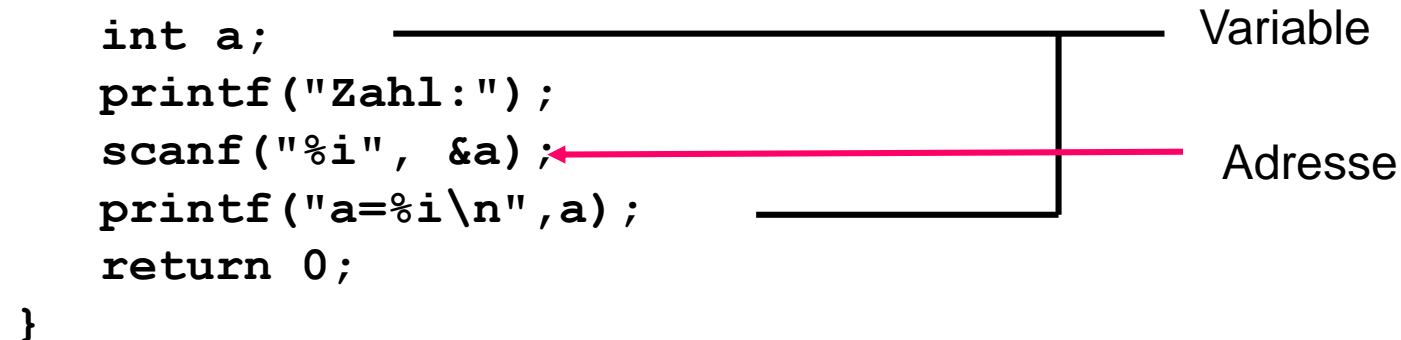
```
int sitzplatz;  
int *ticket;//sitzplatznummer  
ticket=&sitzplatz;
```



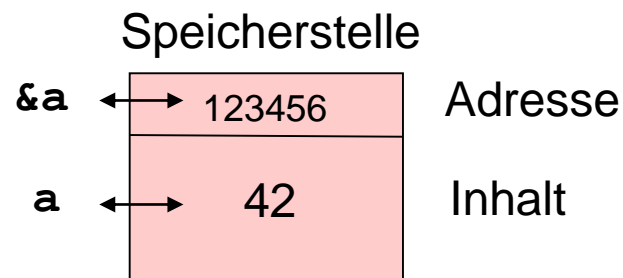
Adressen: Ich weiß, wo du wohnst...!

Jede Variable ist im Rechner unter einer Adresse abgespeichert. In der Regel will man mit dieser Adresse nichts zu tun haben. Aber manchmal doch!

```
int main()  
{  
    int a;  
    printf("Zahl:");  
    scanf("%i", &a);  
    printf("a=%i\n", a);  
    return 0;  
}
```



Es ergibt sich folgender Zusammenhang:



Zeiger (Pointer)

Zeiger sind Variablen, die Adressen speichern. Verwendet man Zeiger, arbeitet man nicht mit den Daten selbst, sondern mit den Adressen, wo die Daten gespeichert sind.

```
int a;  
int *pa;  
a = 42;
```

Adresse einer Variable an eine Zeigervariable übergeben:

```
pa = &a; // pa zeigt auf a
```

Zugriff auf den **Inhalt** einer Adresse:

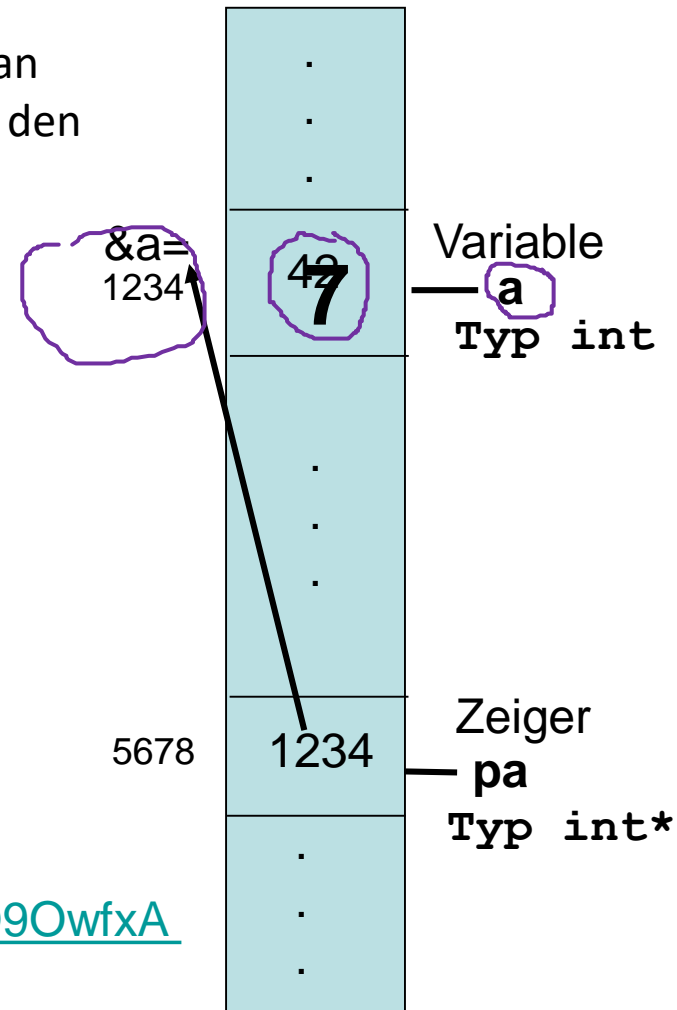
```
*pa = 7;
```



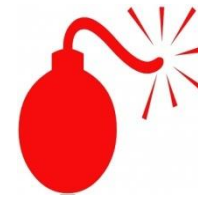
<https://youtu.be/Uvm1O9OwfxA>

Hier ist * ein Operator, der folgendes bedeutet: Inhalt der Adresse, die auf der Zeigervariable gespeichert ist (auf die der Zeiger „zeigt“)

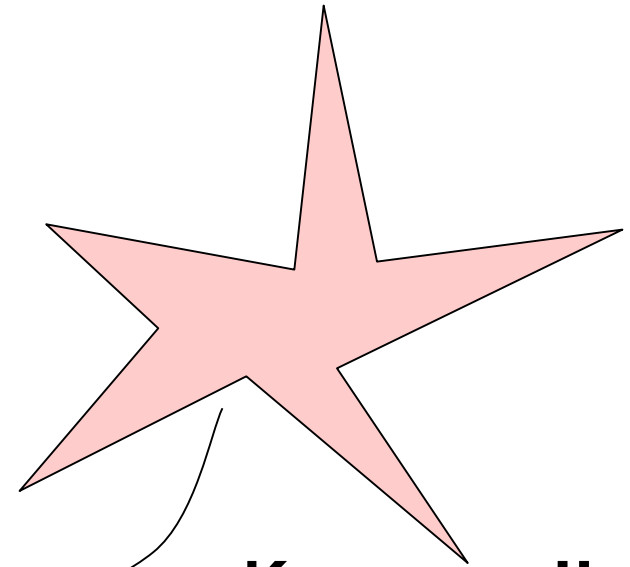
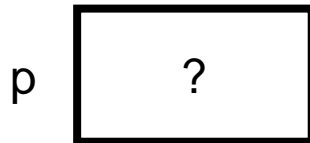
Speicher



Zeiger: Hauptfehler



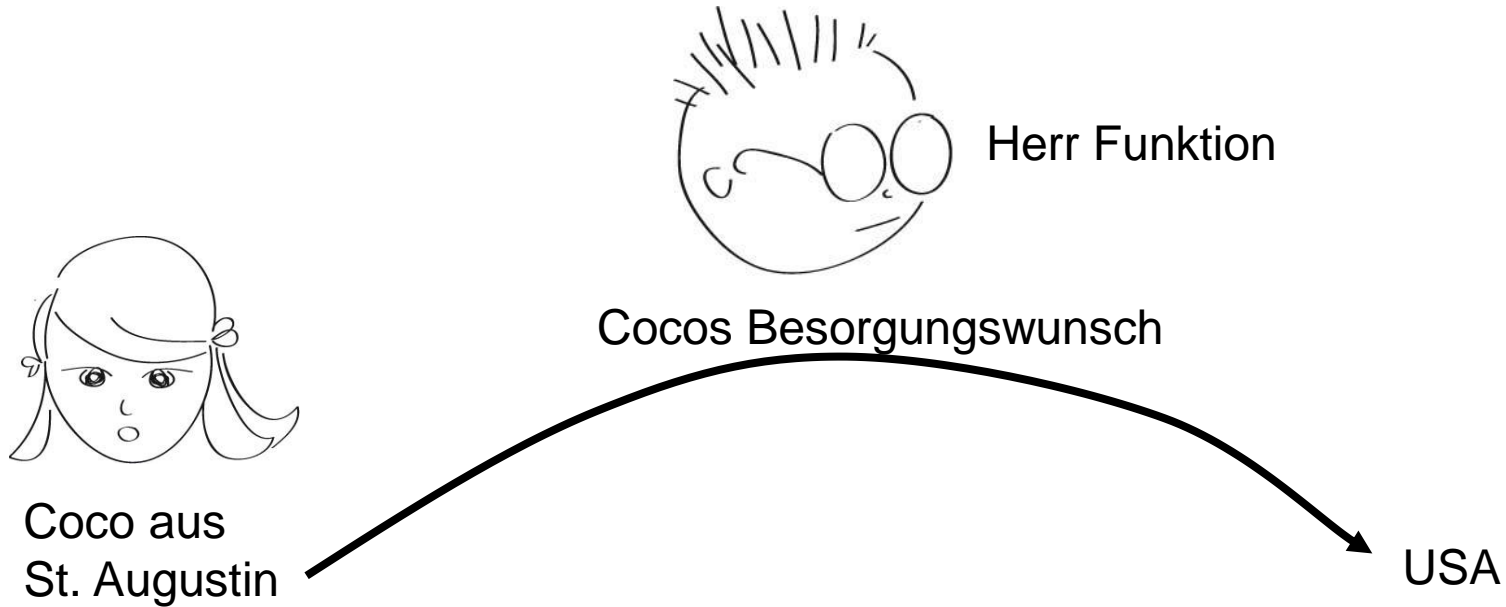
```
int *p;  
*p=13;
```



Krawumm!!

Achtung: Zeigern muss ein Wert (Adresse) zugewiesen werden. Sonst besteht die Gefahr eines Programmabsturzes!!!!!!!!!!!!!!!!!!!!!! Bei Mircocontrollern kann sogar ein Systemabsturz passieren!

Zeiger: Sinnvoller Einsatz

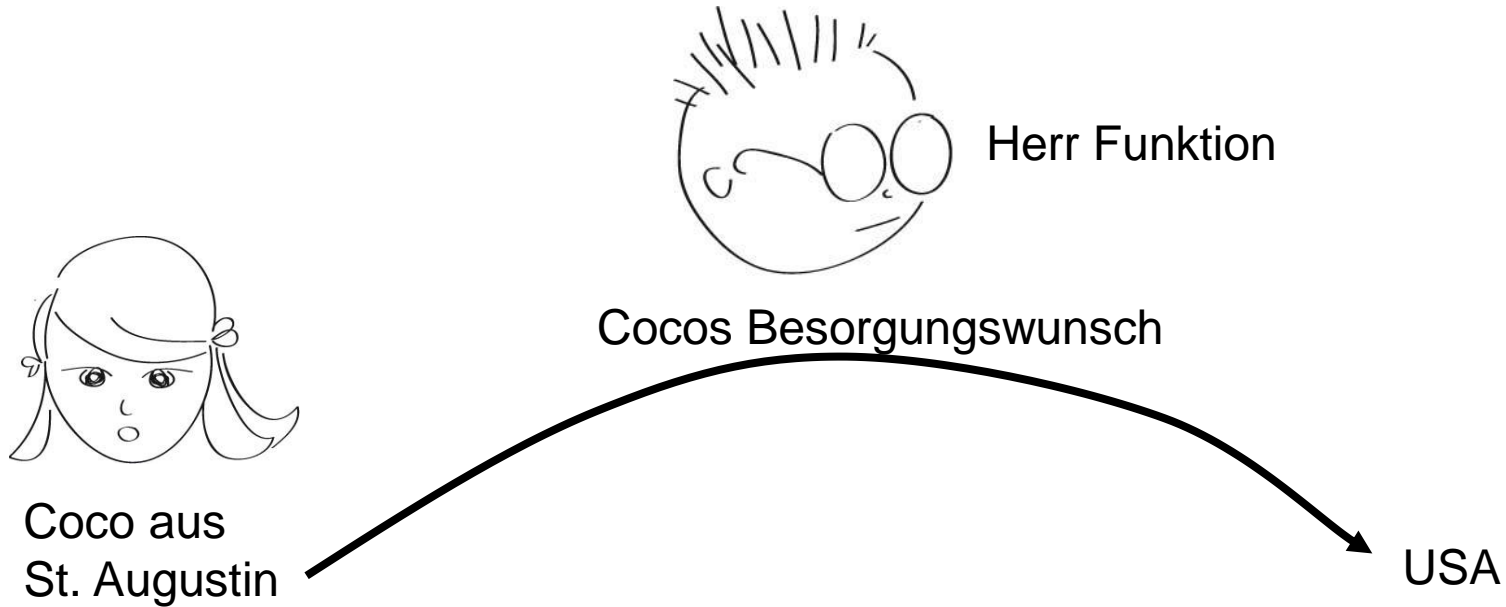


Frage: Wie kommt die Besorgung zu Coco?

Möglichkeit 1: Herr Funktion kommt zurück und bringt die Besorgung zu Coco.

In C: `Uebergabe_der_Besorgung = Herr_Funktion(Besorgung) ;`

Zeiger: Sinnvoller Einsatz

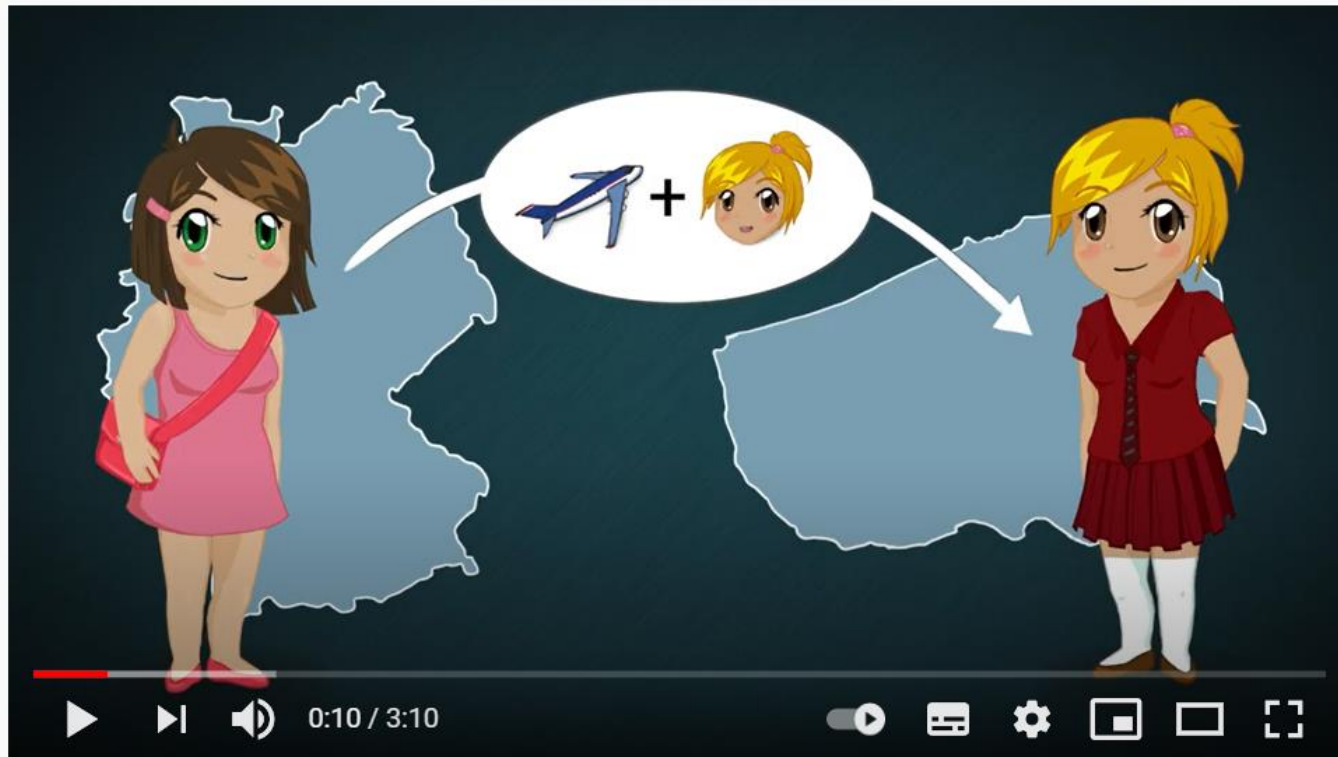


Frage: Wie kommt die Besorgung zu Coco?

Möglichkeit 2: Herr Funktion nimmt die Adresse von Coco mit und schickt ihr die Besorgung, so kann er selbst so lange in den USA bleiben, wie er will.

In C: `Herr_Funktion(&Besorgung) ;`

Zeiger: Sinnvoller Einsatz



Coco startet mit der C Programmierung Teil 5 - Sinnvoller Einsatz von Zeigern



Sinnvoller Einsatz von Zeigern:

<https://www.youtube.com/watch?v=laphh6vagb8>

Zeiger: Noch mal - Sinnvoller Einsatz von Zeigern

Bemerkung: Vielleicht nur sinnvoll, wenn man den Film „Machete“ kennt

• Sie wollen in einem Stadtgebiet eine Wohnung mieten.
• In anderen Worten: Sie wollen von einem Betriebssystem Speicherplatz für Ihr Programm zugewiesen bekommen.
• Zum Glück haben Sie Mister *Compiler* als Anwalt engagiert, er hilft Ihnen dabei und sieht sich mit Ihnen die Wohnungspläne an.

Implementierung eines Programms: `MeineWohnung.c`

```
#include <stdio.h>
int main(){
    int lieblingszahl=42;
    double doppelbett;
    char *schlafzimmer;
    double *bad;
    int *muell;

    return 0;
}
```

1. Eine Lieblingszahl kann man immer gebrauchen.
2. Ihr schönes neues Doppelbett
3. Schlafzimmer
4. Bad
5. Mülltonne

Was haben Zeiger und Machete gemeinsam?



Sinnvoller Einsatz von Zeigern: Machete und Zeiger:

<https://www.youtube.com/watch?v=XPbjBm-HU70>

Funktionen – Arbeit mit Kopien (Call by Value)

Funktionen arbeiten mit Kopien, der an sie übergebenen Werte.

```
void funktion(int a){  
    printf("%i\n",a);  
    a=7;  
    printf("%i\n",a);  
}  
int main() {  
    int a=42;  
    printf("%i\n",a);  
    funktion(a);  
    printf("%i\n",a);  
    return 0;  
}
```

Ausgabe:

42
42
7
42

Die Variable **a** in der Funktion ist nicht die Variable **a** aus **main**. Das sind zwei verschiedene Speicherplätze.



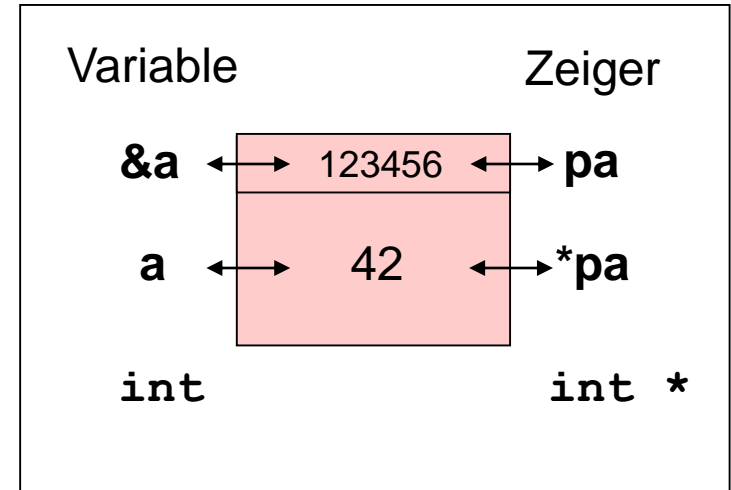
Funktionen – Arbeit mit Adressen (Call by Reference)

Zeiger

```
void funktion(int *pa) {  
    printf("%i\n", *pa);  
    *pa=7;  
    printf("%i\n", *pa);  
}  
  
int main() {  
    int a=42;  
    printf("%i\n", a);  
    funktion(&a);  
    printf("%i\n", a);  
    return 0;  
}
```

Inhaltsoperator

Erinnerung:



Werden an Funktionen Adressen von Variablen übergeben, arbeiten sie mit den Originalwerten. Oben in der Funktion wird also wirklich mit der Variablen **a** aus dem **main** gearbeitet.

Ausgabe:

42
42
7
7



Funktion `tausche1`: so ging es nicht

```
void tausche1(int a, int b){
    int help;
    help = a;
    a  = b;
    b  = help;
    printf("Tausch: a=%i, b=%i\n",a,b);
}

int main() {
    int a = 42, b = 17;
    printf("a=%i, b=%i\n",a,b);
    tausche1(a,b);
    printf("a=%i, b=%i\n",a,b);
    return 0;
}
```

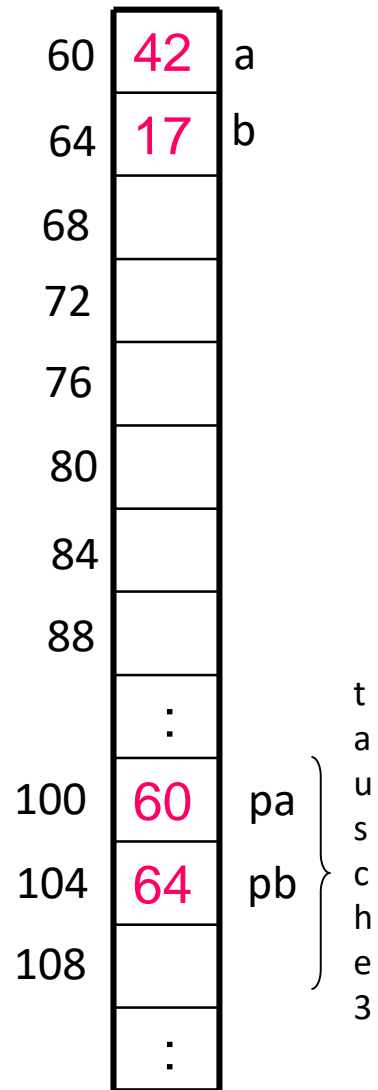
<code>void</code>	<code>tausche1</code>	<code>(int a, int b)</code>
OUT	IN	IN

Mit Zeigern: Funktion `tausche3` 😊

```
void tausche3(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);  
    tausche3(&a, &b);  
    printf("Danach: a=%i, b=%i\n", a, b);  
    return 0;  
}
```

Zeiger

Inhaltsoperator

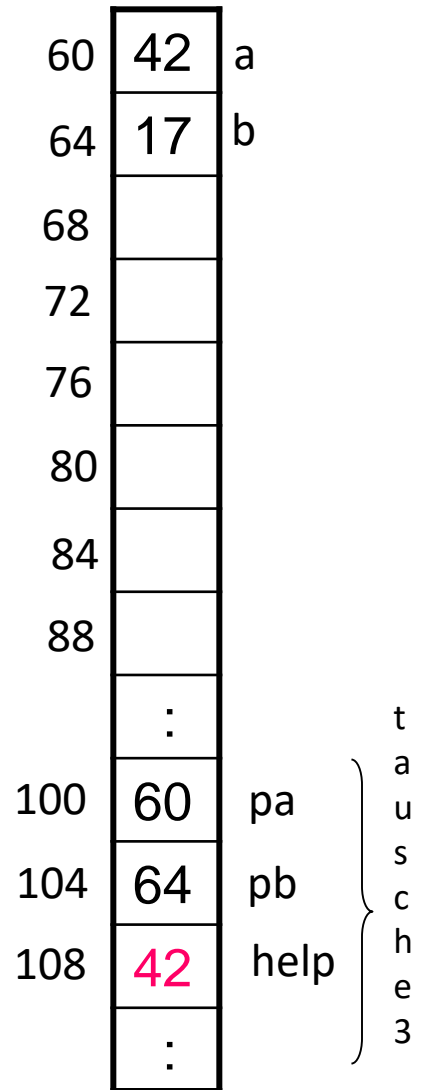


Achtung: Mit einer `return` Anweisung in der Funktion `tausche3` ist dieses Problem nicht zu lösen, weil eine Funktion nur *einen* Wert zurückgeben kann.

Mit Zeigern: Funktion `tausche3`

```
void tausche3(int *pa, int *pb){
    int help;
    help = *pa;
    *pa = *pb;
    *pb = help;
}

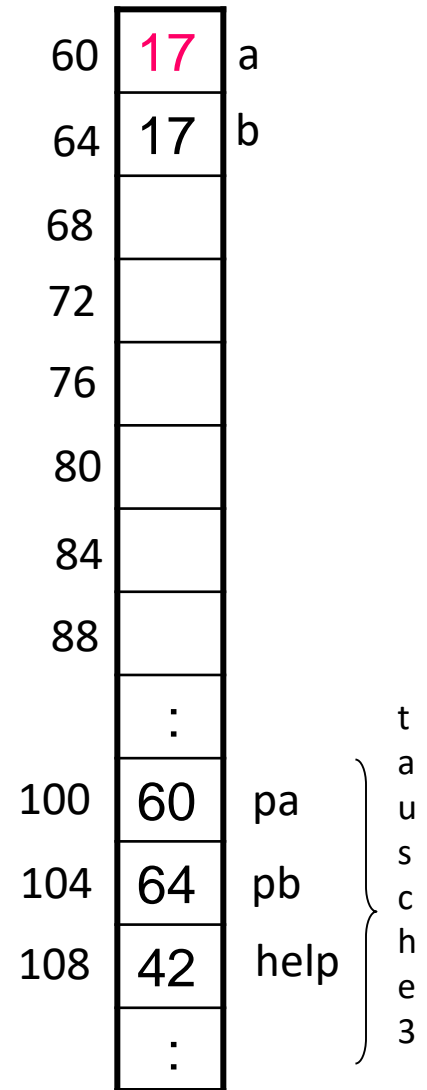
int main(){
    int a = 42, b = 17;
    printf("Davor: a=%i, b=%i\n",a,b);
    tausche3(&a,&b);
    printf("Danach: a=%i, b=%i\n",a,b);
    return 0;
}
```



Mit Zeigern: Funktion `tausche3`

```
void tausche3(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);
    tausche3(&a,&b);
    printf("Danach: a=%i, b=%i\n",a,b);
    return 0;
}
```

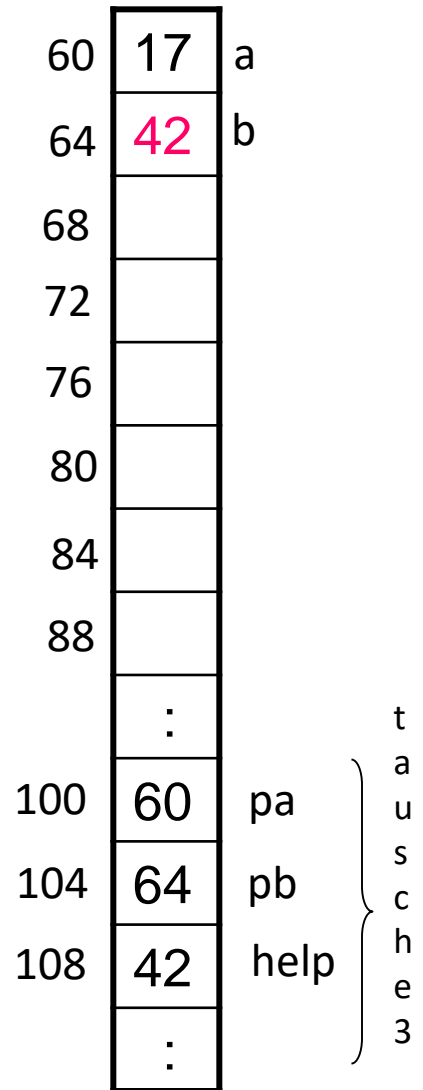


Mit Zeigern: Funktion `tausche3`

```
void tausche3(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);
    tausche3(&a,&b);
    printf("Danach: a=%i, b=%i\n",a,b);
    return 0;
}
```

<code>void</code>	<code>tausche3</code>	<code>(int *pa, int *pb)</code>
<code>OUT</code>	<code>IN-OUT</code>	<code>IN-OUT</code>



Mit Zeigern: noch mal mit Kommentaren

```
//Funktion mit Namen tausche3, die 2 Adressen (die auf ganze
//Zahlen zeigen) uebergeben bekommt und nichts zurueckgibt
void tausche3(int *pa, int *pb){
    int help;//Deklaration einer lokalen Variablen
                //zum Speichern einer ganzen Zahl
    help = *pa;//Zuweisung des Wertes, auf den pa zeigt, auf help
    *pa   = *pb;//an die Stelle, wohin pa zeigt, wird der Wert
                //gespeichert, auf den pb zeigt
    *pb   = help;//an die Stelle, wo pb zeigt, wird der Wert
                //in help gespeichert
}

int main(){
    int a = 42, b = 17;
    printf("Davor: a=%i, b=%i\n",a,b);
    tausche3(&a,&b);//Aufruf der Funktion tausche3 mit 2 Adressen
                    //als Eingabewerte
    printf("Danach: a=%i, b=%i\n",a,b);
    return 0;
}
```



Zeiger: Übung

Frage: Was wird hier ausgegeben? Ihre Vermutung können Sie dann überprüfen, in dem Sie das Programm ausführen lassen.

Achtung: Dafür müssen Sie die Zahlen 1-7 am Anfang der Zeilen löschen.

```
int main() {  
1  int a=42;  
2  int *pa;  
3  pa=&a;  
4  printf("Adresse von a: %i\n", &a) ;  
5  printf("pa=%i\n", pa) ;  
6  printf("Adresse von pa: %i\n", &pa) ;  
7  printf("*pa=%i\n", *pa) ;  
    return 0;  
}
```



Felder und Zeiger: Übung

Was wird hier ausgegeben? ? Ihre Vermutung können Sie dann überprüfen, in dem Sie das Programm ausführen lassen.

Achtung: Dafür müssen Sie die Zahlen 1-11 am Anfang der Zeilen löschen.

```
int main() {  
    1  int feld[5]={0};  
    2  int *pFeld;  
    3  pFeld = &feld[0];  
    4  *pFeld = 17;  
    5  printf("feld[0] = %i\n",feld[0]);  
    6  printf("feld[1] = %i\n",feld[1]);  
    7  printf("&feld[0] = %i\n",&feld[0]);  
    8  printf("&feld[1] = %i\n",&feld[1]);  
    9  printf("feld = %i\n",feld);  
   10  printf("pFeld = %i\n",pFeld);  
   11  printf("*pFeld = %i\n",*pFeld);  
  
    return 0;  
}
```



Zeiger und Funktionen: Beispiele

Call by Value (Arbeit mit Kopien):

```
void f(int a)
```

OUT IN

Call by Value (Arbeit mit Kopien):

```
void tausche1(int a, int b)
```

OUT IN IN

Call by Referenz (Arbeit mit Adressen):

```
void tausche3(int *pa, int *pb)
```

OUT IN-OUT IN-OUT

Gemischt:

```
int beispiel(int *pa, int *pb, int c)
```

OUT IN-OUT IN-OUT IN

Zeiger - Fehlerquellen



- Ein Zeiger ist eine Variable. Aber auf ihnen stehen **IMMER** Adressen.
- Ein Zeiger wird deklariert durch einen Datentyp, der aus einem Grundtyp besteht (char, float, double, int) und einem *. Der Grundtyp gibt den Datentyp an, wohin der Zeiger mal zeigen soll.
- mit einer Variablen vom Datentyp * sind bestimmte Operationen möglich, wie z.B. die Operation *, die als Ergebnis den Inhalt der Adresse des Zeigers hat und ++, womit man eine Adresse weiter springen kann (hierfür muss man den Grundtyp kennen **muss**). Bemerkung: auch in Städten werden Hausnummern=Adressen nur in bestimmten Mindestabständen vergeben!

Achtung: Zeigern **MUSS** ein Wert (Adresse) zugewiesen werden. Sonst besteht die extrem hohe Gefahr eines Computerabsturzes!!!! Wenn nicht sofort, dann eventuell später (was noch schlimmer sein kann).

Bemerkung für später: Zeiger auf Nichts: `int *zeiger=NULL; //gute Lesbarkeit`



Zeiger ...

- „zeigen“ auf andere Variablen, weil sie Adressen anderer Variablen speichern.
- ist eine Variable, auf der eine ganze Zahl steht, die als Adresse interpretiert wird.
- werden deklariert durch Datentyp, der darauf hinweist, was auf der Adresse gespeichert werden kann und *, wodurch klar werden soll, dass es ein Zeiger ist.
- sollte immer ein Wert zugewiesen werden von einer Variablen, die vom Programm vorher angelegt wurde, also <Name einer vorher deklarierten Variablen>
- Zeigeroperationen:
 - & - Adressoperator
 - * - Inhaltsoperator
 - ++ - „eine Adresse weiter“-Operator

Zeiger



Coco startet mit der C Programmierung Teil 4 - Zeiger



Was sind Zeiger: <https://youtu.be/piL5LHhQhM>

Felder und Zeiger: Wiederholung

- Was sind Felder?
 - ➔ Datenstruktur zum Speichern von Daten gleichen Datentyps hintereinander.
- Was ist eine Adresse und was ist der Inhalt einer Variablen?
 - ➔ Adresse: Ort, wo die Variable im Rechnerspeicher steht (Ort wird vom Betriebssystem vergeben),
 - ➔ Inhalt: Wert, der diesem Ort zugewiesen wurde.
- Wozu sind Zeiger gut?
 - ➔ für call by reference-Aufrufe

Zum Merken: z.B. `double *zeiger;`

bedeutet:

`zeiger` ist vom Typ `double *` und
`*zeiger` ist vom Typ `double`