Zeiger

Speicheraufbau

- Variablen haben im Speicher einen Platz an dem sie ihre Werte ablegen können
- Speicher ist eine lineare Abfolge von Bytes
- Jedes Byte hat eine eindeutige Adresse, über die auf den Wert zugegriffen werden kann

Adresse	Inhalt
0x0000000000000000	00000001
0x0000000000000001	10101010
0x0000000000000002	00000001
0x0000000000000003	10101010
0x0000000000000004	10111111
0x0000000000000005	00000001
0x0000000000000006	10101010
0x0000000000000007	00000000
0x0000000000000008	10111111
0x0000000000000009	00000001
0x000000000000000A	10101010
0x000000000000000B	00000000
0x000000000000000C	10111111
0x000000000000000D	00000000
0x000000000000000E	00000000
0x000000000000000F	00000001
0x0000000000000010	10101010
0x0000000000000011	00000000
0x0000000000000012	00000000
0x0000000000000013	10111111
0x0000000000000014	00000000

Speicheraufbau

```
int main() {
    int Variable1;
    int Variable2;
    long int VariableLong;
    char VariableChar;




    return 0;
}
```

Adresse	Inhalt	
0x0000000000000000	00000001	Variable1
0x0000000000000001	10101010	
0x0000000000000002	00000001	
0x0000000000000003	10101010	
0x0000000000000004	10111111	Variable2
0x0000000000000005	00000001	
0x0000000000000006	10101010	
0x0000000000000007	00000000	
0x0000000000000008	10111111	VariableLong
0x0000000000000009	00000001	
0x000000000000000A	10101010	
0x000000000000000B	00000000	
0x000000000000000C	10111111	
0x000000000000000D	00000000	
0x000000000000000E	00000000	
0x000000000000000F	00000001	
0x0000000000000010	10101010	VariableChar
0x0000000000000011	00000000	
0x0000000000000012	00000000	
0x0000000000000013	10111111	
0x0000000000000014	00000000	

Variablen und Speicher

- Intern bei der Übersetzung nach Assembler und MaschinenCode wird nur mit diesen Adressen gearbeitet
- Alle Anweisungen auf MaschinenCode-Ebene adressieren direkt die Speicherzellen, die für eine Variable angelegt wurden
- Können wir das auch aus einem C-Programm heraus?
- Ja, weil C spezielle Funktionen und Datentypen für den Umgang mit Speicheradressen zur Verfügung stellt, sogenannte Zeiger (Pointer)

Adresse einer Variablen

- Der &-Operator liefert die Startadresse einer Variablen im Speicher:

```c
int main() {
    int Variable;

    printf("Speicheradresse: %p \n", &Variable);

    return 0;
}
```

Speicheraufbau

```
int main() {
    int Variable1;
    int Variable2;
    long int VariableLong;
    char VariableChar;




    return 0;
}
```

Adresse	Inhalt	
0x0000000000000000	00000001	Variable1
0x0000000000000001	10101010	
0x0000000000000002	00000001	
0x0000000000000003	10101010	
0x0000000000000004	10111111	Variable2
0x0000000000000005	00000001	
0x0000000000000006	10101010	
0x0000000000000007	00000000	
0x0000000000000008	10111111	VariableLong
0x0000000000000009	00000001	
0x000000000000000A	10101010	
0x000000000000000B	00000000	
0x000000000000000C	10111111	
0x000000000000000D	00000000	
0x000000000000000E	00000000	
0x000000000000000F	00000001	
0x0000000000000010	10101010	VariableChar
0x0000000000000011	00000000	
0x0000000000000012	00000000	
0x0000000000000013	10111111	
0x0000000000000014	00000000	

Beispiel

```c
int main() {
    int Variable1;
    int Variable2;
    long int VariableLong;
    char VariableChar;

    printf("Variable1: %p \n", &Variable1);
    printf("Variable2: %p \n", &Variable2);
    printf("VariableLong: %p \n", &VariableLong);
    printf("VariableChar: %p \n", &VariableChar);

    return 0;
}
```

```
Variable1:    0x7ffeefbff444
Variable2:    0x7ffeefbff440
VariableLong: 0x7ffeefbff438
VariableChar: 0x7ffeefbff437
```

Datentyp Zeiger

- C stellt einen Datentyp Zeiger * zur Verfügung
- Dieser Datentyp ist als Zusatz zu jedem elementaren Datentyp verfügbar, beispielsweise:
  - int * a;
  - short * b;
  - double * c;
  - char * d;
- Variable a enthält nun keinen int-Wert, sondern enthält **eine Adresse** einer Speicherstelle, an der der Wert einer int-Variable abgespeichert ist

Zugriff auf die Daten mit *

- Um den Inhalt auszulesen, auf welchen der Zeiger zeigt, wird der * verwendet (auch **Dereferenz** gennant)

```
int main() {
    int Zahl = 5;
    int Zahl2 = 10;
    int *Zeiger;

    Zeiger = &Zahl;

    Zahl2 = *Zeiger;

    return 0;
}
```

Der Zeiger bekommt die Adresse von Zahl.
Damit zeigt er auf die Variable Zahl

Zahl2 bekommt den Inhalt von Zahl

Beispiel

```c
int main() {
    int Zahl = 5;
    int Zahl2 = 10;
    int *Zeiger = &Zahl;

    *Zeiger = Zahl2;

    printf("%d\n", Zahl);

    return 0;
}
```

Ausgabe: 10

Beispiel

```c
int main() {
    int Zahl = 5;
    int * Zeiger = &Zahl;

    printf("Zahl: %d\n", Zahl);
    printf("Adresse von Zahl: %p\n", &Zahl);
    printf("Zeiger: %p\n", Zeiger);
    printf("Adresse von Zeiger: %p\n", &Zeiger);
    printf("Inhalt auf den der Zeiger zeigt: %d\n", *Zeiger);

    return 0;
}
```

```
Zahl:                                5
Adresse von Zahl:   0x7ffeefbff458
Zeiger:             0x7ffeefbff458
Adresse von Zeiger: 0x7ffeefbff450
Inhalt auf den der Zeiger zeigt: 5
```

Beispiel

```c
int main() {
    int Zahl = 5;
    int * Zeiger = &Zahl;

    printf("Zahl: %d\n", Zahl);
    printf("Adresse von Zahl: %p\n", &Zahl);
    printf("Zeiger: %p\n", Zeiger);
    printf("Adresse von Zeiger: %p\n", &Zeiger);
    printf("Inhalt auf den der Zeiger zeigt: %d\n"

    return 0;
}
```

```
Zahl:                                5
Adresse von Zahl:   0x7ffeefbff458
Zeiger:             0x7ffeefbff458
Adresse von Zeiger: 0x7ffeefbff450
Inhalt auf den der Zeiger zeigt: 5
```

Adresse	Inhalt	
0x00007ffeefbff449	00000001	…
0x00007ffeefbff450	0x58	Zeiger
0x00007ffeefbff451	0xf4	
0x00007ffeefbff452	0xbf	
0x00007ffeefbff453	0xef	
0x00007ffeefbff454	0xfe	
0x00007ffeefbff455	0x7f	
0x00007ffeefbff456	0x00	
0x00007ffeefbff457	0x00	
0x00007ffeefbff458	0x05	Zahl
0x00007ffeefbff459	0x00	
0x00007ffeefbff45A	0x00	
0x00007ffeefbff45B	0x00	
0x00007ffeefbff45C	00000000	…

Deklaration von Zeigern

```
1  #include <stdio.h>
2
3  int main() {
4      int* Zahl1, Zahl2;
5      int Zahl3 = 4;
6
7      Zahl2 = Zahl3;
8      Zahl1 = Zahl3;
9
10
11     return 0;
12 }
13
```

Incompatible integer to pointer conversion assigning to 'int *' from 'int'; take the address with &

Insert '&'

Fix

Deklaration von Zeigern

```c
int main() {
    int *Zeiger1, *Zeiger2;
    int Zahl = 4;

    Zeiger1 = &Zahl;
    Zeiger2 = &Zahl;

    *Zeiger1 = 4;
    *Zeiger2 += 1;
    (*Zeiger2)++;

    printf("%d\n", Zahl);

    return 0;
}
```

Ausgabe: 6

Datentypen Zeiger

```c
int main() {
    int intZahl = 1;
    float floatZahl = 4.3;
    long int longZahl = 0xff;
    int *intPtr;
    float *floatPtr;
    long int *longPtr;

    intPtr = &intZahl;
    floatPtr = &floatZahl;
    longPtr = &longZahl;

    *intPtr += 3;
    *floatPtr += 3.33;
    *longPtr += 0xabc;

    printf("%d\n", intZahl);
    printf("%f\n", floatZahl);
    printf("%lx\n", longZahl);

    return 0;
}
```

Datentypen Zeiger

```
int main() {
    int intZahl = 1, *intPtr = &intZahl;
    float floatZahl = 4.3, *floatPtr = &floatZahl;
    long int longZahl = 0xff, *longPtr = &longZahl;

    *intPtr += 3;
    *floatPtr += 3.33;
    *longPtr += 0xabc;

    printf("%d\n", intZahl);
    printf("%f\n", floatZahl);
    printf("%lx\n", longZahl);

    return 0;
}
```

void

- void kann bei Funktionen genutzt werden, welche keinen Wert zurückgeben sollen
- Typ der Daten ist mit void nicht angegeben (-> void * untypisierter Zeiger)
- Es ist nicht möglich eine Variable mit void zu definieren
- Bei main() ist der Typ int im Standard definiert, meist funktioniert jedoch auch void (hier ist der Rückgabewert dann undefiniert)

```
void addiere(int s1, int s2, double *summe) {
     *summe = s1 + s2;
     return;  /* formal erlaubt aber ohne jegliche Wirkung, d.h. kann entfallen */
}
```

void Pointer

- void Zeiger müssen vor der Verwendung in den Typ konvertiert werden:

```
#include <stdio.h>

int main() {
    int Zahl = 4;
    void *Zeiger;

    Zeiger = &Zahl;

    printf("Zahl: %d\n", *(int *)Zeiger);


    return 0;
}
```

Funktionsparamter

- Durch Verwendung von Zeigern auf Variablen als formale Parameter einer Funktion kann man Seiteneffekte erzeugen

- Möchte man dies nutzen, um Parameter in ihrem Wert zu verändern, ist das bei den Datentypen in der Parameterliste anzugeben:

```
void func(int *a);
```

- Beim Aufruf muss für jeden Parameter wieder der Datentyp stimmen, also:

```
int i;
func(&i);
```

Zeiger als Parameter – Beispiel

```c
#include <stdio.h>

void division(int dividend, int divisor, int *quotient, int *remainder){
    *quotient = dividend / divisor;
    *remainder = dividend % divisor;
}

int main() {
    int Zahl1 = 21;
    int Zahl2 = 4;
    int Ergebnis;
    int Rest;

    division(Zahl1, Zahl2, &Ergebnis, &Rest);

    printf("Rechnung: %d / %d = %d (Rest: %d) \n", Zahl1, Zahl2, Ergebnis, Rest);

    return 0;
}
```

Zeiger als Parameter – Beispiel

```c
#include <stdio.h>

void division(int dividend, int divisor, int *quotient, int *remainder){
    *quotient = dividend / divisor;
    *remainder = dividend % divisor;
}

int main() {
    int Zahl1 = 21;
    int Zahl2 = 4;
    int Ergebnis;
    int Rest;

    division(Zahl1, Zahl2, &Ergebnis, &Rest);

    printf("Rechnung: %d / %d = %d (Rest: %d) \n", Zahl1, Zahl2, Ergebnis, Rest);

    return 0;
}
```

Da der "quotient" und der "remainder" Pointer sind, wird das Ergebnis direkt in die Variablen "Ergebnis" und "Rest" geschrieben. Die Pointer “zeigen” auf die Variablen (enthalten deren Adressen)

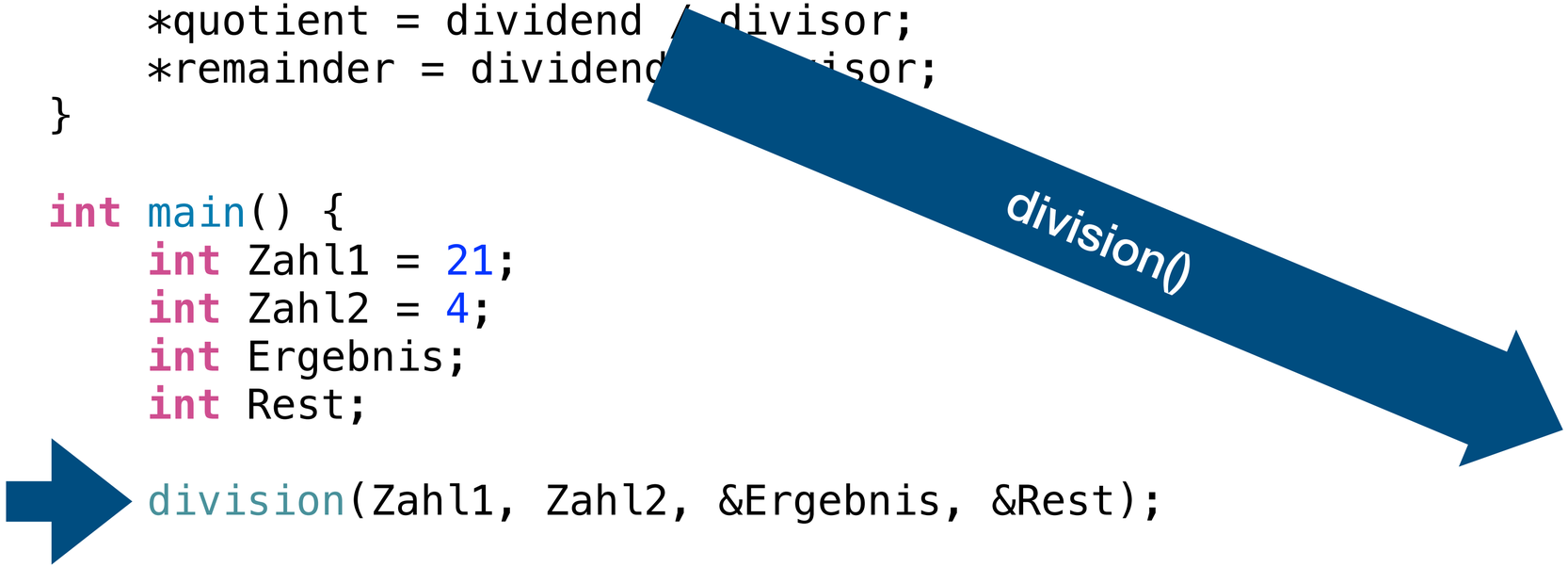
Zeiger als Parameter - Ablauf

```c
#include <stdio.h>

void division(int dividend, int divisor, int *quotient, int *remainder){
    *quotient = dividend / divisor;
    *remainder = dividend % divisor;
}

int main() {
    int Zahl1 = 21;
    int Zahl2 = 4;
    int Ergebnis;
    int Rest;

    division(Zahl1, Zahl2, &Ergebnis, &Rest);

    printf("Rechnung: %d / %d = %d (Rest: %d) \n", Zahl1, Zahl2, Ergebnis, Rest);

    return 0;
}
```

Startadresse	Inhalt	
0x00	21	Zahl1
0x04	4	Zahl2
0x08	undefined	Ergebnis
0x0C	undefined	Rest

Zeiger als Parameter - Ablauf

```c
#include <stdio.h>

void division(int dividend, int divisor, int *quotient, int *remainder){
    *quotient = dividend / divisor;
    *remainder = dividend % divisor;
}

int main() {
    int Zahl1 = 21;
    int Zahl2 = 4;
    int Ergebnis;
    int Rest;

    division(Zahl1, Zahl2, &Ergebnis, &Rest);

    printf("Rechnung: %d / %d = %d (Rest: %d) \n", Zahl1, Zahl2, Ergebnis, Rest);

    return 0;
}
```



Startadresse	Inhalt	
0x00	21	Zahl1
0x04	4	Zahl2
0x08	undefined	Ergebnis
0x0C	undefined	Rest
0x10	21	dividend
0x14	4	divisor
0x18	0x08	quotient
0x20	0x0C	remainder

Zeiger als Parameter - Ablauf

```c
#include <stdio.h>

void division(int dividend, int divisor, int *quotient, int *remainder){
    *quotient = dividend / divisor;
    *remainder = dividend % divisor;
}

int main() {
    int Zahl1 = 21;
    int Zahl2 = 4;
    int Ergebnis;
    int Rest;

    division(Zahl1, Zahl2, &Ergebnis, &Rest);

    printf("Rechnung: %d / %d = %d (Rest: %d) \n", Zahl1, Zahl2, Ergebnis, Rest);

    return 0;
}
```

Startadresse	Inhalt	
0x00	21	Zahl1
0x04	4	Zahl2
0x08	**5**	Ergebnis
0x0C	**1**	Rest
0x10	21	dividend
0x14	4	divisor
0x18	0x08	quotient
0x20	0x0C	remainder

Eingabe in C – scanf

- Eingaben in C erfolgen mittels der Funktion scanf(), die folgendes Aufrufschema hat (Signatur in stdio.h):

```
int eingabe1, eingabe2;
scanf(„%d %d“, &eingabe1, &eingabe2);
```

- Da Funktionen nur einen Rückgabewert haben, hier aber potentiell mehrere Werte zurückgegeben werden müssen, übergeben wir scanf einen Zeiger auf die Variablen, die zu füllen sind

- scanf führt dann im Prinzip das folgende durch:

```
*iPtr = wert1; // wert1 wurde eingegeben

*jPtr = wert2; // wert2 wurde eingegeben
```

Zeiger auf Zeiger auf Zeiger auf …

- Zeiger sind auch nur Variablen, die einen Speicherplatz besitzen
- Möchte man diesen Adressieren, kann ebenso der &-Operator eingesetzt werden, also:

```
int *iPtr;
iPtrPtr = &iPtr;
```

- Welchen Datentyp muss iPtrPtr dann haben?
- Er muss ein Zeiger auf einen int-Zeiger (int *) sein:

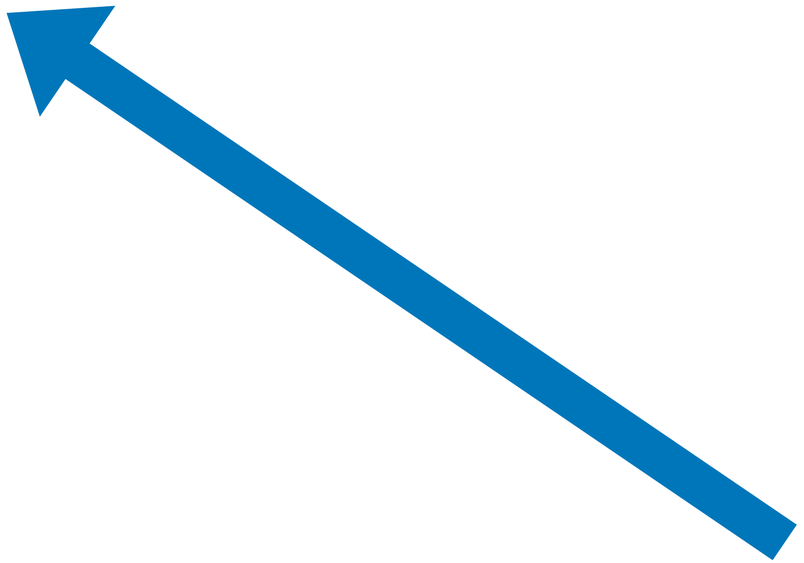
```
int **iPtrPtr;
```

- Damit können auch Zeiger per Seiteneffekt aus Funktionen heraus mit neuen Zeigerwerten versehen werden:

```
*iPtrPtr = &j;
```

Zeigerarithmetik

```c
int main() {
    int Zahl1 = 21;
    int Zahl2 = 4;
    int Zahl3 = 12;
    int *Zeiger;

    Zeiger = &Zahl3;

    Zeiger += 2;

    printf("Zahl: %d \n", *Zeiger);

    return 0;
}
```



Zahl: 21

Zeigerarithmetik

```c
int main() {
    int Zahl1 = 21;
    int Zahl2 = 4;
    int Zahl3 = 12;
    int *Zeiger;

    Zeiger = &Zahl3;

    Zeiger += 2;

    printf("Zahl: %d \n", *Zeiger);

    return 0;
}
```

- Zeiger enthalten Adressen, mit welchen man rechnen kann
- Eine Addition von 1 bedeutet, dass die Adresse um **sizeof(Datentyp)** erhöht wird
- Nur Additionen und Subtraktionen erlaubt
- Sehr fehleranfällig!

```
Zahl: 21
```

Zusammenfassung Zeiger

- Zeiger erlauben viele nützliche Dinge
- Es gibt kaum ein C-Programm ohne die Verwendung von Zeigern•
- Man kann mit ihnen alles machen, insbesondere vieles falsch
- Zugriffe auf Speicherbereiche außerhalb des lokal für das Programm definierten Bereiches führen zum Absturz ihres Programmes, im Extremfall reißt es das ganze Betriebssystem mit weg
- Man kann davon ausgehen, dass eine Vielzahl aller Abstürze kommerzieller Programme auf fehlerhafte Zeigerverwendung zurück zu führen ist