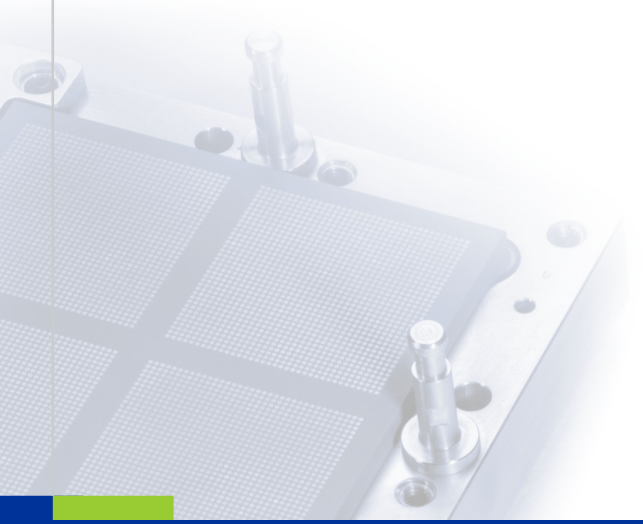
R|R|Z|N

Programmierung mit C

Zeiger



Zeiger (Pointer)

- ... ist eine Variable, die die Adresse eines Speicherbereichs enthält. Der Speicherbereich kann
  - ... kann den Wert einer Variablen enthalten oder
  - ... dynamisch allokiert sein.
- ... verweist auf einen bestimmten Bereich im Speicher.
- ... enthält eine Referenz auf eine andere Variable.
- Zeiger werden genutzt, um
  - ... Speicherbereiche dynamisch zu verwalten.
  - ... Adressen von Datenobjekten direkt an Funktionen zu übergeben.
  - ... Funktionen als Argumente an andere Funktionen zu übergeben.
  - ... Datenstrukturen wie Listen, Bäume, Stacks und Queues zu implementieren.
- Zeiger
  - ... führen dazu, dass die Programmstruktur häufig unübersichtlich wird.
  - ... sind die häufigste Fehlerquelle in C.
  - ... erfordern Disziplin vom Programmierer.

Zeiger deklarieren

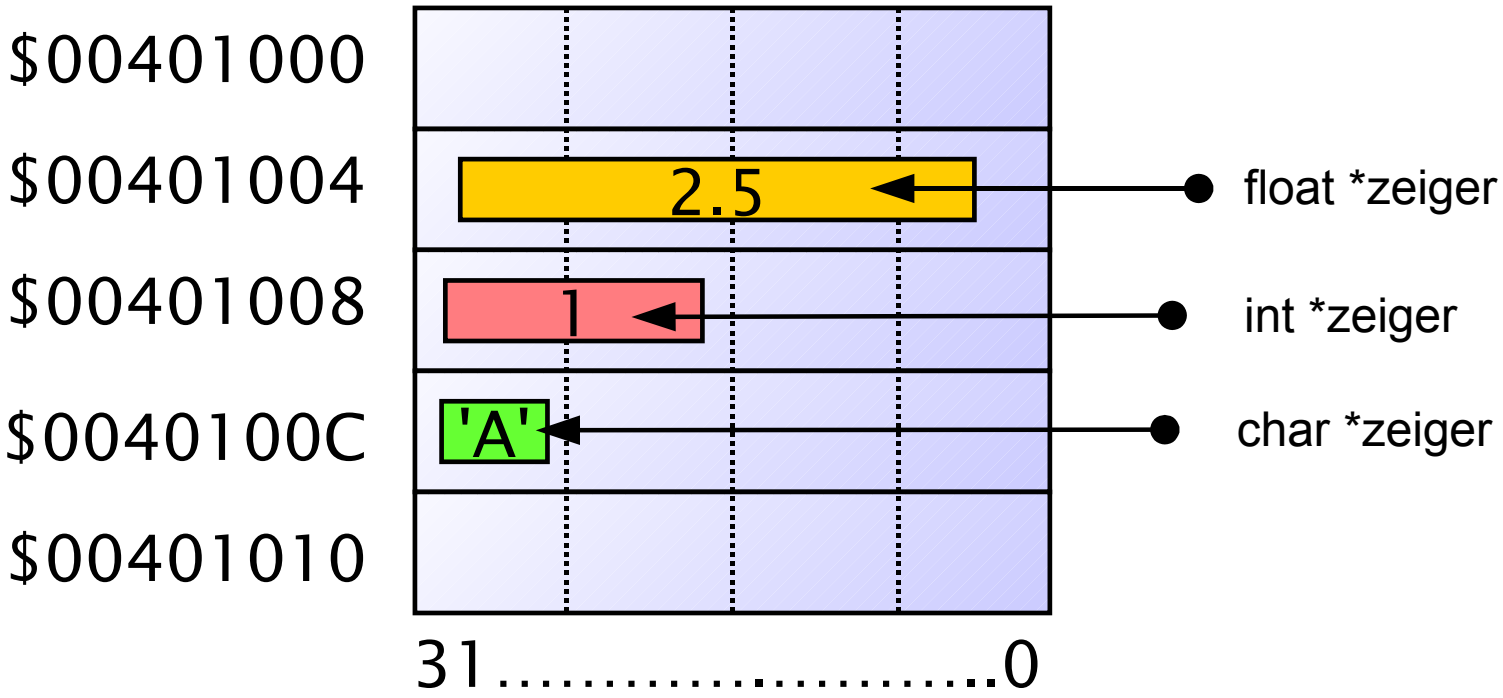
R|R|Z|N

```
Datentyp *Name_der_Zeigervariablen
```

- Jeder Datentyp benötigt eine bestimmte Menge an Speicher.
  - Der Zeiger zeigt auf den Anfang des definierten Speicherbereichs.
  - Der Zeiger wird mit demselben Datentyp vereinbart wie die Variable, auf die er zeigt.
  - Wichtig für Zeigerarithmetik!
- Das Sternchen gekennzeichnet die Variable als Zeiger.
- Der Variablenname ist eindeutig und spiegelt die Nutzung der Variablen wieder.

Beispiel

```
int main(void)
{
   int *zeigerInt;
   char *zeigerChar;
   float *zeigerFloat;

   /* ptrInt ist als Zeiger deklariert. */
   /* ptrGanz ist eine Variable */
   int *ptrInt, ptrGanz;
}
```


$00401000
$00401004
$00401008
$0040100C
$00401010
2.5
1
'A'
float *zeiger
int *zeiger
char *zeiger
31.....0

```
int *ptrXpoint, *ptrYpoint, *ptrZpoint, ptrPoint;
```

- Hier wird jede Variable als Zeiger gekennzeichnet.
- Der Programmier kann auf einen Blick sehen, welche Variablen als Zeiger genutzt werden oder nicht.

```
int* ptrXpoint, ptrYpoint, ptrZpoint, ptrPoint;
```

- `ptrXpoint` wird als Zeiger deklariert.
- Alle anderen Namen sind Synonyme für Variablen.

```
int * ptrXpoint;
int*ptrYpoint;
```

- Leerzeichen werden vom Compiler ignoriert.
- Ein Zeiger wird gelesen als "Der Inhalt, auf den der Zeiger verweist, ist ein int.". Der Name des Zeigers symbolisiert den Verweis. Aus diesem Grund sollte das Sternchen (Asterisk) direkt vor dem Namen stehen.

Zeiger initialisieren

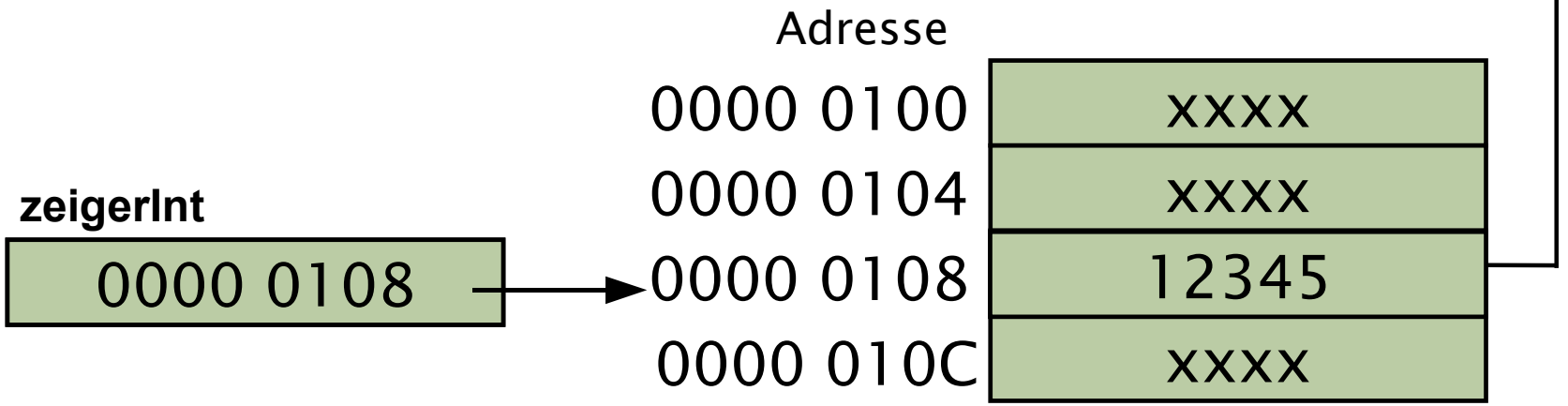
R|R|Z|N

```
Name_der_Zeigervariablen  = &Variable;
```

- Der Zeiger wird mit der Adresse einer vorher bekannten Variablen initialisiert.
- Der Zeiger besitzt als Wert die Adresse der Variablen, auf die er zeigt.
- Das kaufmännische UND-Zeichen wird als Adressoperator bezeichnet. Es wird die Adresse der Variablen an den Zeiger übergeben und nicht der Wert der Variablen. Zwischen dem kaufmännischen Zeichen und dem Variablennamen darf kein Leerzeichen stehen.
- Wenn einem Zeiger keine Adresse, sondern ein Wert zugewiesen wird, wird eine Warnung ausgegeben.

```
int main(void)
{
  int *zeigerInt;
  int summe;

   summe = 12345;
   zeigerInt = &summe;
}
```

Speicherort der Variablen summe

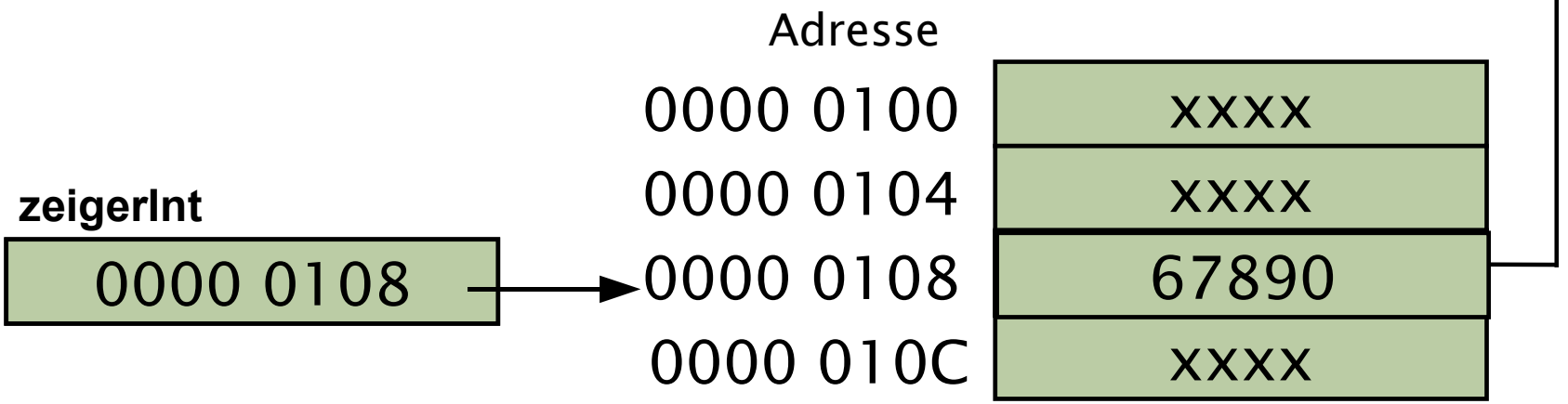


```
*Zeigervariable  = Ausdruck;
```

- Das Sternchen wird als Dereferenzierungsoperator bezeichnet.
- Mit Hilfe des Dereferenzierungsoperator wird der Wert verändert, auf den der Zeiger zeigt.
- Der Zeiger besitzt als Wert eine Adresse.
  - An dieser Adresse ist ein Datenwort hinterlegt.
  - Das Datenwort an dieser Adresse wird mit Hilfe des Dereferenzierungsoperators und des Zeigers verändert.
  - Der Wert des Zeigers selber wird nicht verändert.
- Wenn Sie einen Zeiger innerhalb eines Ausdrucks nutzen, klammern Sie die Dereferenzierung!

```
int main(void)
{
  int *zeigerInt;
  int summe;

   summe = 12345;
   zeigerInt = &summe;
   *zeigerInt = 67890;
}
```

Speicherort der Variablen summe



Die Adresse einer Variablen ausgeben

R|R|Z|N

```
int main(void)
{
   int wert;
   wert = 5;

   printf("Der Wert: %d\n", wert);

   printf("Die Adresse: %p\n", &wert);

   return 0;
}
```

In diesem Fall wird der Wert der Variablen wert als Integer auf dem Bildschirm ausgegeben.

Mit Hilfe des Formatzeichens %p wird eine Adresse auf dem Bildschirm ausgegeben.

Wenn der Adressoperator dem Variablennamen voran gestellt ist, wird die Adresse der Variablen ausgegeben.

NULL-Zeiger

R|R|Z|N

- Ein Null-Zeiger zeigt auf keine gültige Adresse und ist mit 0 initialisiert.
- Die Verwendung der Null für unbenutzte Zeiger ist eine Konvention der Programmierer!
- Es gibt verschiedene Möglichkeiten einen NULL-Zeiger zu erzeugen.
  - `char *ptr = 0;`
    Dem Zeiger wird eine 0 zugewiesen.
  - `#define NULL (void *) 0`
    `int *ptr = NULL;`
    Es wird eine Konstante definiert, die einen Zeiger mit 0 initialisiert.
  - `#include <stdef.h>`
    `int *ptr =NULL;`
    Es wird die vorgefertigte Konstante aus der Bibliothek <*stdef.h*> genutzt.

NULL-Zeiger

R|R|Z|N

```
#include <stdio.h>
#define NULL (void *) 0
#include <stdlib.h>
int main(void)
{
   int *pointer;
   pointer = NULL;

   if (NULL == pointer) {
     printf("Keine gueltige Adresse\n");
   }
   else {
     *pointer = 3;
   }
    system("Pause");
    return 0;
}
```

Hier wird die vordefinierte Konstante NULL als Makro definiert.

Warum wird die Reihenfolge Konstante == Variable gewählt?

void-Zeiger

- ... werden als untypisierte oder generische Zeiger bezeichnet.
- Ein Zeiger vom Datentyp `void` ist ein typenloser Zeiger.
- ... ist zu jedem Datentyp kompatibel.
- ... werden genutzt, wenn die Größe des Speicherbereichs nicht bekannt ist.
- ... kann eine beliebige Adresse zugewiesen werden.

Beispiel


```
#include <stdio.h>

int main(void)
{
   void *pointer;
   int wert;

   wert = 10;

   pointer = (int *)&wert;

   *(int *)pointer = 100;

   printf("Pointer = %p , %d", pointer, wert);
   return 0;
}
```

Hier findet eine Typumwandlung eines Zeigers in der Form `pointer = (Datentyp *)&Variable` statt.

Der Zeiger wird mit einem Verweis auf eine Variable vom Datentyp Integer initialisiert.

Einen void-Zeiger kann jeder andere Zeigertyp zugewiesen werden.

Beispiel

```
#include <stdio.h>

int main(void)
{
   void *pointer;
   int wert;

   wert = 10;

   pointer = (int *)&wert;

   *(int *)pointer = 100;

   printf("Pointer = %p , %d", pointer, wert);
   return 0;
}
```

Hier findet eine Dereferenzierung eines void-Zeigers in der Form `*(Datentyp *)pointer = Ausdruck;` statt.

Der void-Zeiger kann nicht direkt dereferenziert werden. Zuerst wird ein temporärer Zeiger vom Datentyp int erzeugt. Dieser temporärer Zeiger wird dereferenziert.

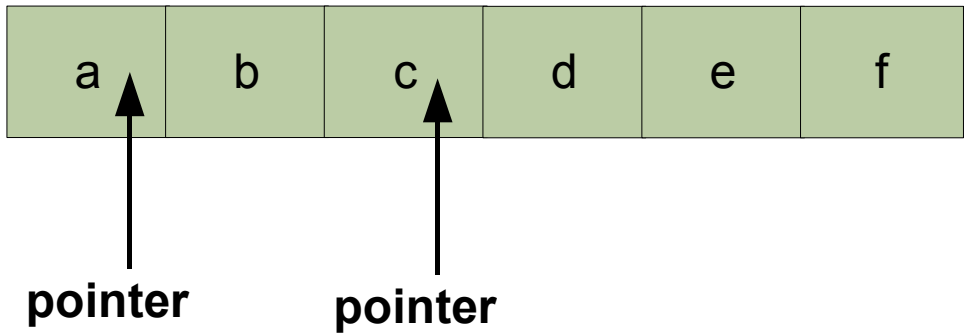
Zeiger-Arithmetik

- Folgende Operationen können mit Zeigern ausgeführt werden:
  - Addition von Ganzzahlwerten.
  - Subtraktion von Ganzzahlwerten.
  - Inkrementieren.
  - Dekrementieren.
- Ein Zeiger wird immer um ein Vielfaches der Speichergröße seines Datentyps erhöht oder erniedrigt.
- Das Arbeiten mit Vergleichsoperatoren ist erlaubt.
- Wird häufig bei Arrays eingesetzt.

```
int main(void)
{
    int *pointer;
    int wert;

    pointer = &wert;
    pointer += 10;
    pointer--;

    return 0;
}
```

Beispiel

```
int main()
{
    int *pointer;
    int feld[] = {'a', 'b', 'c', 'd', 'e', 'f'};

    pointer = &feld[0];
    pointer += 2;
    return 0;
}
```



Zeiger einem anderen Zeiger zuweisen

R|R|Z|N

- Mit Hilfe des Zuweisungsoperator wird einem Zeiger der Wert eines anderen Zeigers zugewiesen.
- Beide Zeiger zeigen auf die gleiche Adresse und können den Wert an dieser Adresse verändern.
- Wenn der Wert eines Zeigers an einen anderen Zeiger zugewiesen wird, wird kein Adressoperator benötigt.

```
int main(void)
{
   int *pointer;
   int *zeiger;
   int wert;

   pointer = &wert;
   zeiger = pointer;
   *zeiger = 5;
   *pointer = *zeiger * 2;

   return 0;
}
```

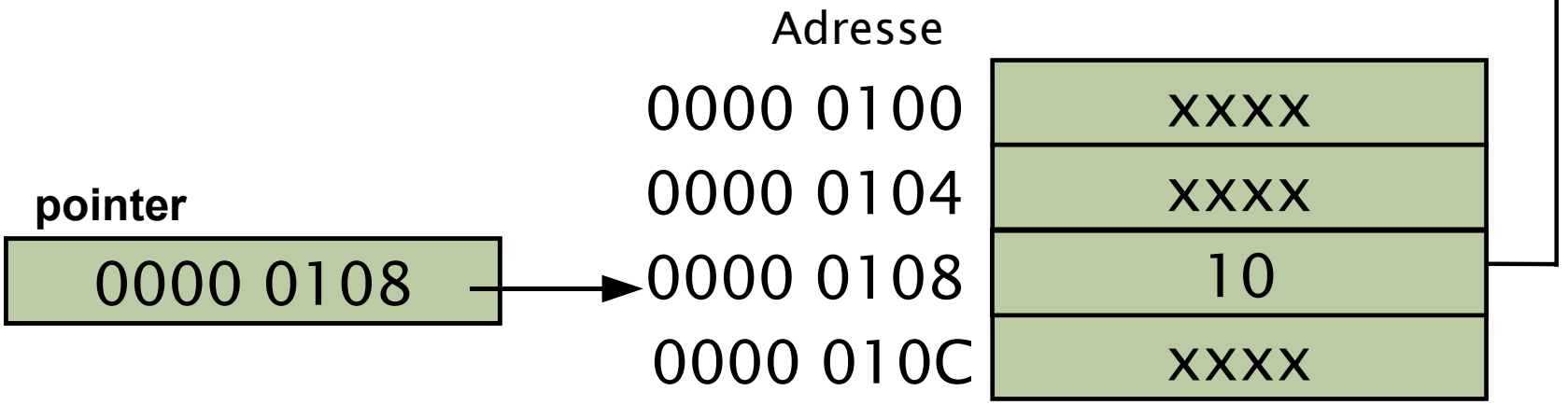
Zeiger auf Zeiger

- Ein Zeiger zeigt auf einen Zeiger, der wiederum auf den Wert einer Variablen zeigt.
- Mehrfach-Indirektion.
- ... werden für die Matrizenberechnung eingesetzt.
- Ein Zeiger auf Zeiger wird folgendermaßen deklariert:
  `datentyp **Name;`
- Mit Hilfe von `Zeiger_Zeiger = &Zeiger;` wird dem Zeiger die Adresse eines anderen Zeigers zugewiesen.
- Die Anweisung `**Zeiger_Zeiger = ausdruck;` verändert den Wert an der Adresse, die in dem Zeiger abgelegt ist, auf den der Zeiger zeigt.
- Die Anweisung `*Zeiger_Zeiger = wert;` würde dem Zeiger auf Zeiger eine neue Adresse zuweisen, die einen beliebigen Wert besitzt.

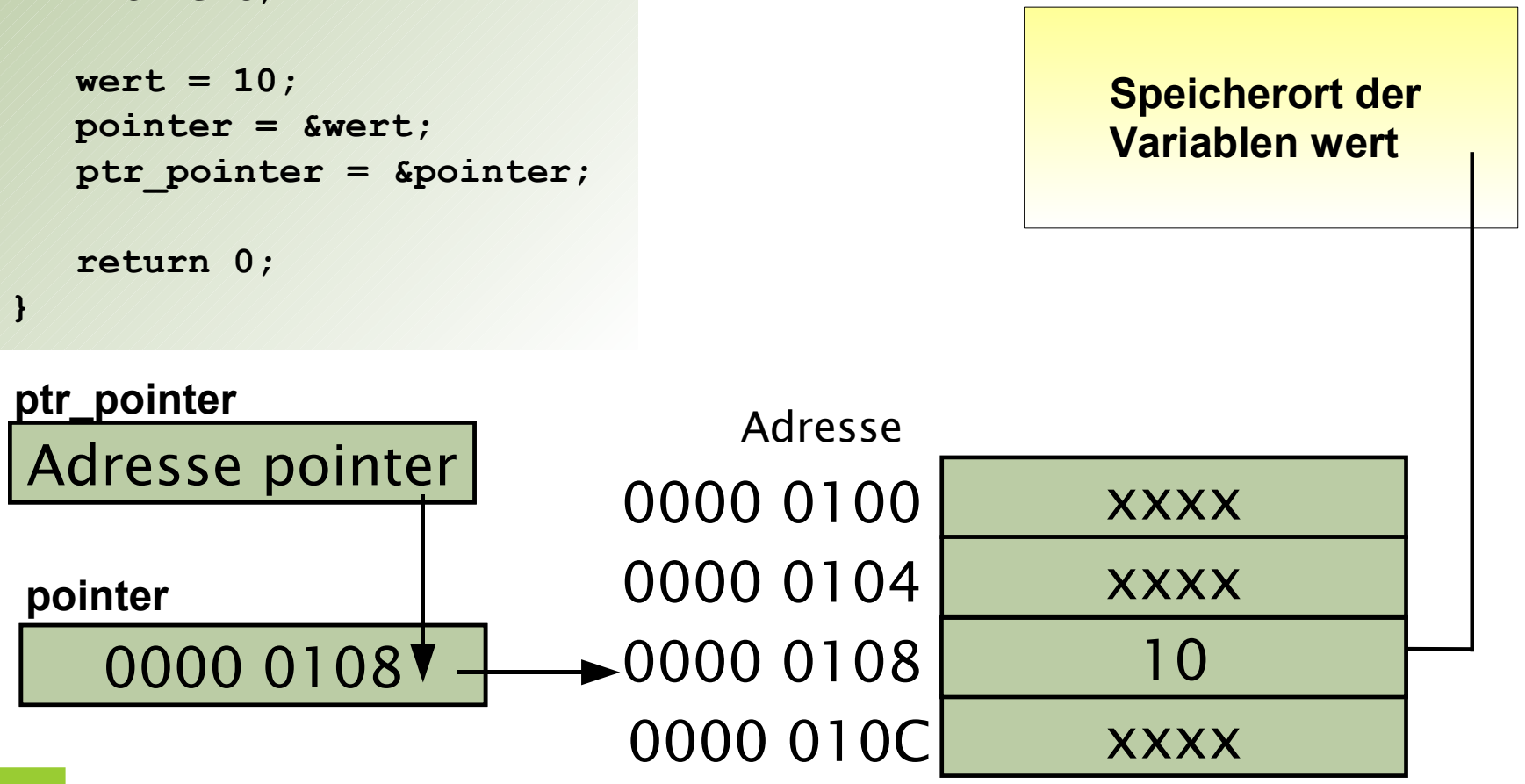
Beispiel

```
int main(void)
{
    int *pointer;
    int wert;

    wert = 10;
    pointer = &wert;


    return 0;
}
```

Speicherort der Variablen wert



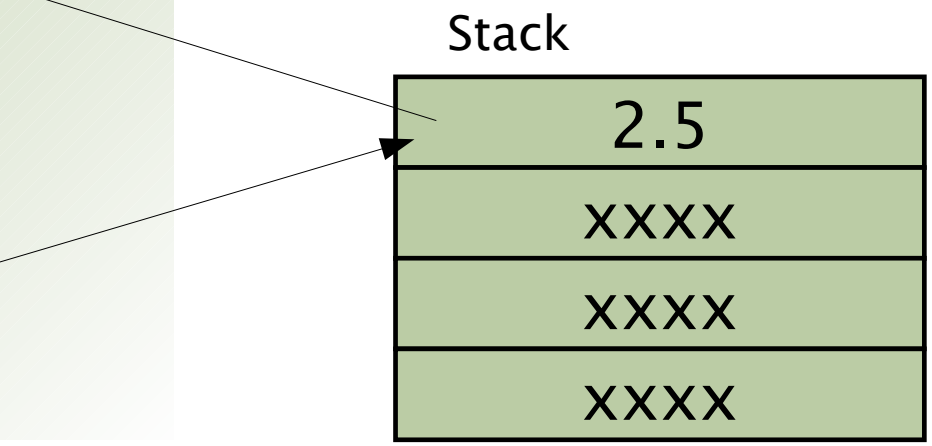
```
int main(void) {
   int *pointer;
   int **ptr_pointer;
   int wert;

   wert = 10;
   pointer = &wert;
   ptr_pointer = &pointer;

   return 0;
}
```

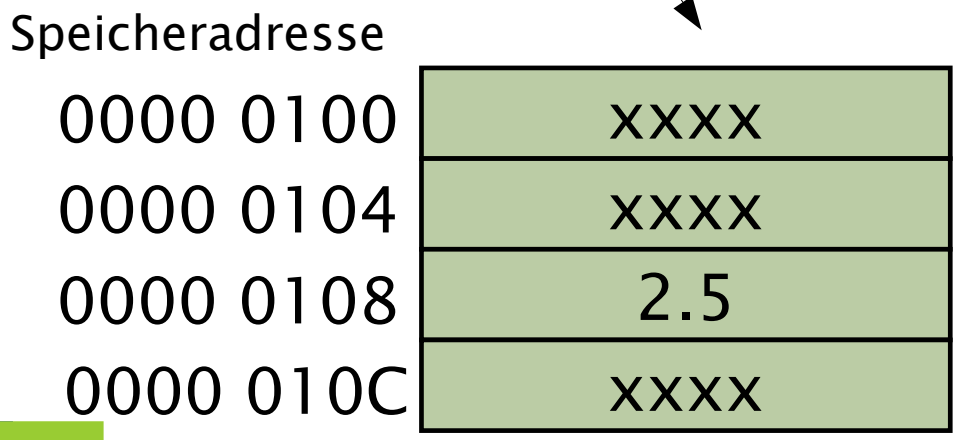


Übergabe von Werten als Argument einer Funktion

R|R|Z|N

```
void malZwei(float zahl ) {
  zahl = (zahl) * 2;
}

int main(void) {
   float wert;
   wert = 2.5
   malZwei(wert);
   return 0;
}
```





Übergabe von Adressen als Argument einer Funktion


```
void malZwei(float *zahl ) {
  *zahl = (*zahl) * 2;
}

int main(void)
{
   float wert;
   wert = 2.5

   malZwei(&wert);

   return 0;
}
```

Übergabe von Adressen als Argument einer Funktion


```
void malZwei(float *zahl ) {
  *zahl = (*zahl) * 2;
}

int main(void)
{
   float wert;
   wert = 2.5

   malZwei(&wert);

   return 0;
}
```

Der Funktion wird ein Zeiger als Argument übergeben. Der Zeiger sollte auf Werte vom Datentyp float zeigen.

Mit Hilfe des Zeigers kann die Funktion Werte an der angegebenen Adresse verändern.

Bedenken Sie, das der Zugriff auf die Adresse auch ungewollte Änderungen erlaubt.

Übergabe von Adressen als Argument einer Funktion


```
void malZwei(float *zahl ) {
  *zahl = (*zahl) * 2;
}

int main(void)
{
   float wert;
   wert = 2.5

   malZwei(&wert);

   return 0;
}
```

Mit Hilfe der Dereferenzierung wird der Wert an der Adresse, auf die der Zeiger verweist, verändert.

Hier wird der Wert an der angegebenen Adresse mal zwei genommen und anschließend das Ergebnis an der angegebenen Adresse gespeichert.

Übergabe von Adressen als Argument einer Funktion

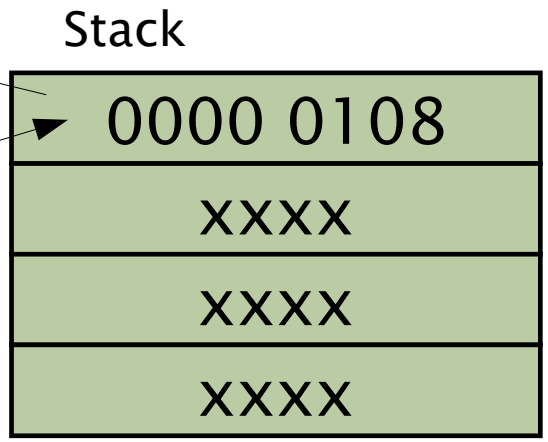

```
void malZwei(float *zahl ) {
  *zahl = (*zahl) * 2;
}

int main(void)
{
   float wert;
   wert = 2.5

   malZwei(&wert);

   return 0;
}
```

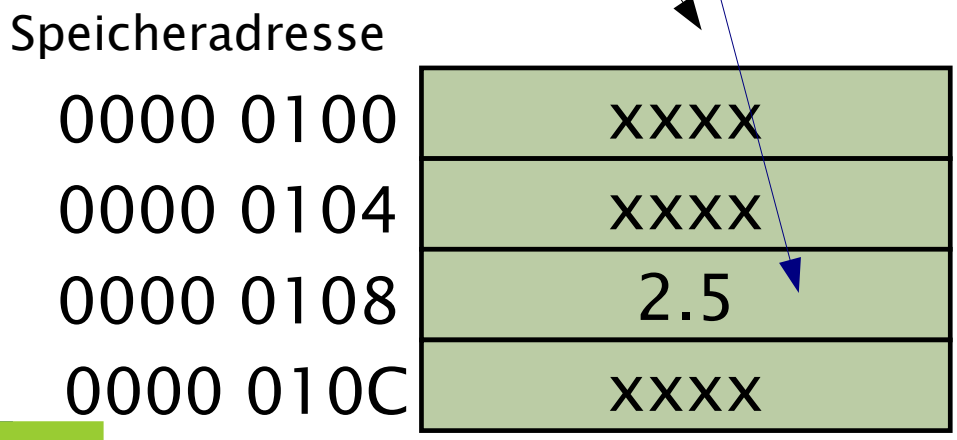
Der Funktion wird die Adresse der Variablen wert übergeben.

Der Variablennamen beginnt mit dem Adressoperator. Hier wird nicht der Wert der Variablen übergeben, sondern die Adresse der Variablen.

Übergabe von Adressen als Argument einer Funktion


```
void malZwei(float *zahl ) {
  *zahl = (*zahl) * 2;
}

int main(void) {
   float wert;
   wert = 2.5
   malZwei(&wert);
   return 0;
}
```





Funktionsadresse

- Eine Funktion hat eine physikalische Adresse im Speicher, die einem Zeiger zugewiesen werden kann. Diese Adresse ist der Eingangspunkt der Funktion und wird beim Aufruf der Funktion genutzt.
- Wenn ein Zeiger auf diese Adresse zeigt, kann die Funktion darüber aufgerufen werden.
- Der Funktionsname ohne Klammern oder Argumenten enthält als Wert die Adresse im Speicher.

Funktionszeiger

```
Funktiontyp (*Zeiger)(Argument1, ..., ArgumentN);
```

- ... können Funktionen als Argumente an andere Funktionen übergeben werden.
- Der Zeiger verweist auf eine Funktion von einem bestimmten Datentyp
- Funktionstyp ist abhängig vom Datentyp (Rückgabewert) der Funktion.
- Die Deklaration des Zeigers muss aufgrund der Prioritäten von Operatoren immer geklammert werden.
- In Klammern steht die Anzahl der Argumente und deren Datentyp entsprechend der Funktion, auf die der Zeiger zeigt.

Beispiel


```
#include <stdio.h>

int Antwort(int zahl) {
   return zahl;
}

int main(){
   int (*ptr)(int) = Antwort;

   printf("%d\n", ptr(3));

   return 0;
}
```

Konstante Zeiger

- `const Datentyp  *Pointer`
  - Zeiger auf einen konstanten Wert.
  - Der Zeiger selber ist nicht konstant und kann verändert werden.
- `Datentyp *const Pointer`
  - Zeiger auf einen variablen Wert.
  - Der Zeiger selber ist konstant und kann nicht verändert werden.
- `const Datentyp *const Pointer`
  - Zeiger zeigt auf einen konstanten Wert.
  - Der Zeiger ist konstant.
  - Diese Art wird sehr selten genutzt.

Beispiel

```
const char constChar = 'A';
char nonConstChar = 'a';

const char *ptrConstWert;
char *const ptrConstZeiger = &nonConstChar;

ptrConstWert = &constChar;

/* Fehler: Veränderung einer Konstante */
*ptrConstWert = 'B';

ptrConstWert = &nonConstChar;

/* Fehler: Veränderung einer Konstante */
*ptrConstWert = 'B';
```

Beispiel

```
const char constChar = 'A';
char nonConstChar = 'a';

const char *ptrConstWert;
char *const ptrConstZeiger = &nonConstChar;


*ptrConstWert++;  /* Zeiger nicht konstant */

/* Fehler: Konstanter Zeiger */
ptrConstZeiger = &ConstChar;

/* Verweis auf einen variablen Wert */
*ptrConstZeiger = 'B';
```

Ein Zitat von Howard und LeBlanc (2003)

- Zeiger (engl. Pointer) gehören zu den fehleranfälligsten Bereichen moderner Programmiertechniken. Etliche moderne Sprachen wie Java, C# und Visual Basic stellen daher gar keinen Datentyp für Zeiger zur Verfügung.
- Der Einsatz von Zeigern ist von Haus aus kompliziert und erfordert von Ihnen ein exzellentes Verständnis der Speicherverwaltung Ihres Compilers.
- Viele verbreitete Sicherheitsprobleme, insbesondere Pufferüberläufe, lassen sich auf Fehler bei der Benutzung von Zeigern zurückführen.

Zitat aus „Das C-Lösungsbuch“; Clovis L. Tondo, Scot E. Gimpel

Quelle: /5/

Mögliche Fehler

- Bei der Deklaration eines Zeigers, wird nur der Zeiger angelegt, aber nicht das Objekt worauf er zeigt.
- Zeiger müssen initialisiert werden, andernfalls können Datenbereiche
  - ... des eigenen Programms oder
  - ... anderer Programm oder
  - ... des Betriebssystems überschrieben werden.
- Folgende Zuweisung erzeugt keinen Compiler-Fehler:

```
int *pointer;
*pointer = 100;
```