



Zeiger/Pointer in C



DI Thomas Helml

SEW

SJ 2018/19



Inhaltsverzeichnis

- ① Pointer
 - ① Call-by-Reference
 - ① NULL-Zeiger
 - ① Zeiger und Arrays
 - ① Zeigerarithmetik
 - ① Zeiger als Rückgabewert
-



Pointer

- ① Zeiger = **Adresse** + **Typ** eines Objekts
 - ① sprich: Zeiger referenziert (zeigt auf) Adresse
 - ① Typ des Objekts gibt an, wie groß die Speicherzelle ist
 - ① sowohl für Lese- und Schreiboperation
 - ① Sprechweise (abhängig vom Typ):
 - ① Zeiger auf `int` oder
 - ① `int`-Zeiger
-



Pointer

① Deklaration

`Datentyp *name;`

① `name = Bezeichner`

① gleiche Namensregeln wie bei Variablen

① Typ der Variable ist **Datentyp** *

① z.B. `int *name;`

① `// Datentyp: int *`



Pointer

❶ Beispiel:

```
int *p;
```

❷ Datentyp: `int *`

❸ d.h. in `p` kann die Adresse eines `int`-Wertes gespeichert werden

❹ Adressoperator `&`

❶ Liefert Adresse einer Variablen

❷ `&p` => Adresse von `p`



Pointer

① VOR der Verwendung eines Zeigers muss dieser auf eine Stelle im Speicher zeigen

① Beispiel:

```
int *ptr;           // Zeiger auf int
```

```
int value = 123; // eine int-Variable
```

```
ptr = &value;      // der Zeiger ptr zeigt auf value
```



Pointer

```
int *ptr;           // Zeiger auf int
int value = 123;    // eine int-Variable

ptr = &value;       // der Zeiger ptr zeigt auf value
```

Bezeichner	Adresse	Wert
value	0xbfe5d3bc	123
ptr	0xbfe5d3c0	0xbfe5d3bc

A red line originates from the memory address '0xbfe5d3bc' in the 'ptr' row, extends horizontally to the right, then vertically upwards, and finally horizontally to the left, ending with an arrowhead pointing to the memory address '0xbfe5d3bc' in the 'value' row. This visualizes the pointer 'ptr' storing the address of the 'value' variable.



Pointer

① Spezielles Formatzeichen für Adressen: %p

```
int main ()
{
    int *ptr;
    int value = 255;
    ptr = &value;

    printf („Adresse ptr: %p\\n“, &ptr);
    printf („zeigt auf : %p\\n“, ptr);
    printf („Adresse value: %p\\n“, &value);
    printf („Wert value: %d\\n“, value);
}
```

```
Adresse ptr: 0xbfe5d3c0
zeigt auf: 0xbfe5d3bc
Adresse value: 0xbfe5d3bc
Wert value: 255
```



Verweisoperator *

- ① Achtung! Verweisoperator ist
 - ① ein unärer Operator
 - ① ungleich binärer, arithmetischer Operator für Multiplikation *
 - ① `ptr` = Zeiger
 - ① `*ptr` = Objekt, auf das `ptr` zeigt
-



Pointer

❶ Beispiel:

```
int x, y, *ptr;    // ptr ist ein Zeiger auf ein int
ptr = &x;          // ptr zeigt auf Adresse von x
y = *ptr;          // Variable y wird das Objekt,
                  // auf das ptr zeigt, zugewiesen
```

❷ Die Zuweisung $y=x$ würde dasselbe machen

❸ `*ptr` ist wie eine Variable zu verwenden

Typ von ptr	int * (Zeiger auf int)
Typ von *ptr	int



Pointer

IT Beispiel

```
int a, b, *pa;
```

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

```
*pa = 12; // a wird der Wert 12 zugewiesen
```

```
*pa += 2; // a wird um 2 erhöht
```

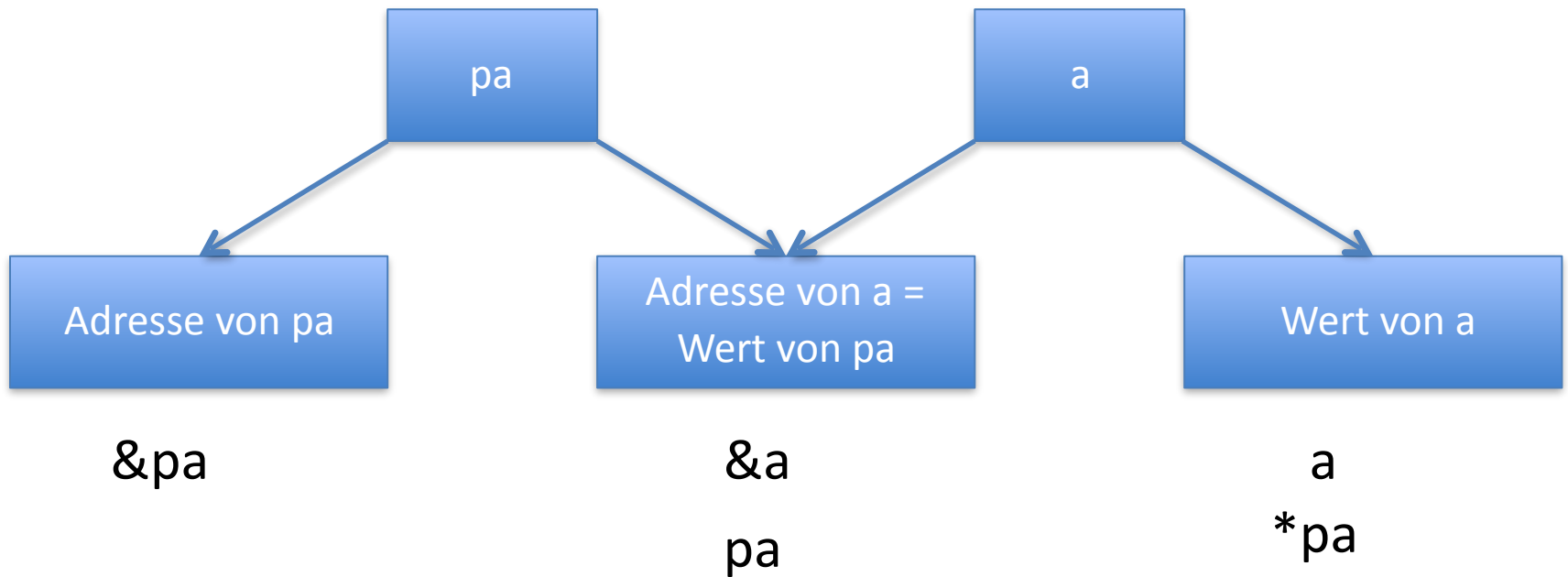
```
b = a*2;
```

```
// Wert von a=14, b=28
```



Pointer

`pa = &a;`





Call-by-Reference

- ① Per default werden Funktionsparameter in C call-by-Value übergeben
 - ① d.h. eine Kopie der Variable
 - ① Änderungen der übergebenen Variablen haben keinen Einfluss auf die außerhalb der Funktion
 - ① Call-by-Reference
 - ① Zeiger auf die Variablen werden übergeben
 - ① bei Änderung => Auswirkung auf Variable
-



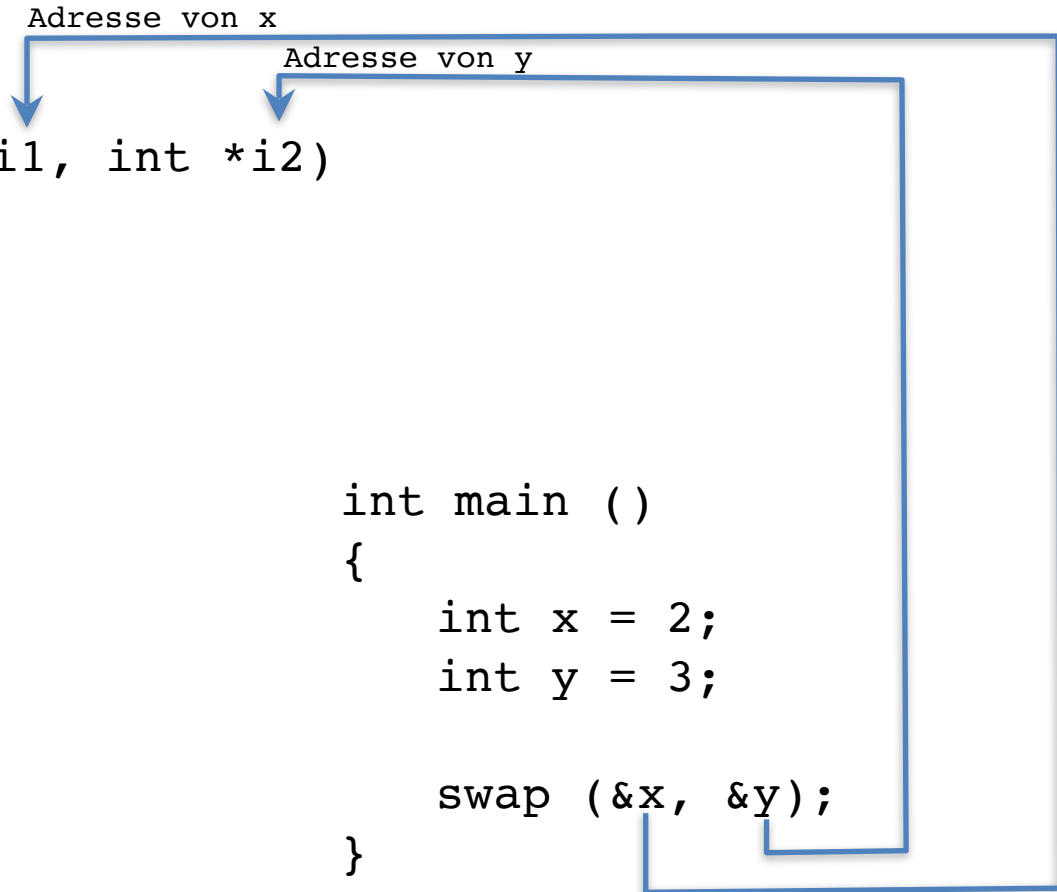
Call-by-Reference

IT Beispiel

```
void swap (int *i1, int *i2)
{
    int help;
    help = *i1;
    *i1 = *i2;
    *i2 = help;
}
```

```
int main ()
{
    int x = 2;
    int y = 3;

    swap (&x, &y);
}
```





NULL-Zeiger

- ① Verweisoperator darf bei Zeiger mit gültiger Adresse verwendet werden
 - ① ansonsten: Segmentation fault!
(Speicherzugriffsverletzung)
 - ① DAHER:
 - ① Zeiger mit NULL initialisieren
 - ① VOR Zugriff überprüfen, ob Zeiger == NULL gilt
-



NULL-Zeiger

Beispiel

```
int main ()
{
    int *iptr = NULL; // Zeiger mit NULL initialisieren

    // Überprüfung vor der ersten Verwendung
    if (iptr == NULL)
    {
        printf("Zeiger hat keine gültige Adresse");
        return -1;
    }
    // iptr kann verwendet werden ...
    return 0;
}
```



NULL-Zeiger

IT Beispiel

```
int main ()
{
    // Zeiger mit NULL initialisieren
    int *iptr1 = NULL;
    int *iptr2 = NULL;
    int ival1, ival2;

    //Initialisierung: Zeiger erhält Adresse von ival1
    iptr1 = &ival1;
    // ival1 erhält den Wert 123
    *iptr1 = 123;

    iptr2 = &ival2;
    *iptr2 = 456;

    iptr2 = iptr1;
    *iptr2 = 456;

    printf („*iptr1: %d“, *iptr1);
    printf („*iptr2: %d“, *iptr2);
    printf („ival1: %d“, ival1);
    printf („ival2: %d“, ival2);

    return 0;
}
```

Ausgabe?



Zeiger und Arrays

① Gegeben: Array a

```
int a[4] = {10, 20, 30, 40};
```

① In C:

① a ist konstanter Zeiger auf 1. Array-Element a[0]

① Somit kann a einem Zeiger zugewiesen werden

```
int *pa;
```

```
pa = a;
```

① pa zeigt auf a[0]

① pa+1 zeigt auf a[1]



Zeiger und Arrays

```
int a[4] = {10, 20, 30, 40};  
int *pa;  
pa = a;
```

Zeiger			Speicher		Werte			
a	→ pa	→	0xa0	10	a[0]	pa[0]	*(a+0)	*(pa+0)
a+1	→ pa+1	→	0xa3	20	a[1]	pa[1]	*(a+1)	*(pa+1)
a+2	→ pa+2	→	0xa7	30	a[2]	pa[2]	*(a+2)	*(pa+2)
a+3	→ pa+3	→	0xab	40	a[3]	pa[3]	*(a+3)	*(pa+3)



Zeiger und Arrays

Beispiel

```
int main ()
{
    int a[4] = {10, 20, 30, 40};
    int *pa;
    pa = a;
    for (i = 0; i<4; i++)
        printf („Adresse: %p, Wert: %2d\\n“,
                pa+i, *(pa+i));
    return 0;
}
```



Zeigerarithmetik

- ① arithmetische Operationen (+, -, ++, --) und Vergleiche sind in C erlaubt

```
int i=3, anzahl = 0;  
int x, a[10], *pa;
```

```
pa = a;
```

- ① `pa + i` zeigt auf `a[i]`

- ① `pa = pa + i;`

- ① Zeiger `pa` wird „versetzt“
-



Zeigerarithmetik

- ① Operator ++ und -- sind erlaubt, ABER
 - ① erhöhen immer Zeiger und nicht Inhalt!
 - ① *pv++ entspricht * (pv++)
- ① Addition zweier Zeiger ist erlaubt (sinnlos)
- ① Subtraktion liefert Anzahl der Array Elemente zwischen Zeigern

```
pa = a+3;
```

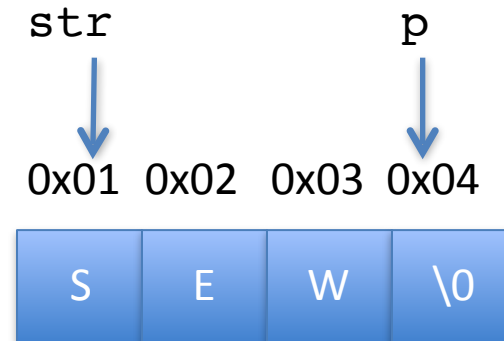
```
anzahl = pa - a; // Anzahl bekommt den Wert 3
```



Zeigerarithmetik

IT Beispiel strlen

```
int strlen(char *str) {  
    char *p;  
    p = str;  
    while (*p != '\\0')  
        p++;  
    return (p-str);  
}
```



$$p - str = 0x04 - 0x01 = 0x03 = 3$$



Zeigerarithmetik

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

```
int main()
```

```
{
```

```
    int v[] = {10, 20, 30, 40, 50};
```

```
    int i, *pv;
```

```
    for (pv = v; pv <= v + 4; pv++)
```

```
        printf (" *pv = %d  ", *pv);
```

AUSGABE?

```
// *pv = 10 *pv = 20 *pv = 40 *pv = 50
```

```
    return 0;
```

```
}
```



Zeigerarithmetik

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

```
int main()
```

```
{
```

```
    int v[] = {10, 20, 30, 40, 50};
```

```
    int i, *pv;
```

```
    for (pv = v, i = 1; i<=4; i++)
```

```
        printf (" pv[i] = %d", pv[i]);
```

AUSGABE?

```
// pv[i]=20 pv[i]=30 pv[i]=40 pv[i]=50
```

```
    return 0;
```

```
}
```



Zeigerarithmetik

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

```
int main()
```

```
{
```

```
    int v[] = {10, 20, 30, 40, 50};
```

```
    int i, *pv;
```

```
    pv = v;
```

```
    i = 0;
```

```
    do {
```

```
        printf ("*(pv+i) = %d ", *(pv+i));
```

```
        i++; pv++;
```

```
    }while (pv + i <= &v[4]);
```

```
// *(pv+i) = 10 *(pv+i) = 30 *(pv+i) = 50
```

```
    return 0;
```

```
}
```

AUSGABE?



Zeigerarithmetik

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

```
int main()
```

```
{
```

```
    int v[] = {10, 20, 30, 40, 50};
```

```
    int i, *pv;
```

```
    for (pv = v + 4; pv >= v; pv --)
```

```
        printf (" v[%d] = %d ", pv - v, v[pv-v]);
```

AUSGABE?

```
// v[4]=50 v[3]=40 v[2]=30 v[1]=20 v[0]=10
```

```
    return 0;
```

```
}
```



Zeiger als Rückgabewert

- ① Funktionen, die Zeiger zurück geben, liefern nur Anfangsadresse des Rückgabewertes

```
Typ *Funktionsname (Parameter) {}
```

- ① Verwendung primär bei

- ① Arrays

- ① Strings

- ① Strukturen

- ① als Rückgabewert
-



Zeiger als Rückgabewert

```
#define MAX 255
```

```
char buf[MAX] = "";
```

```
char *strsearch (char *str, char ch) {  
    char *pChar = str;
```

```
    while (*pChar!='\0') {  
        if (*pChar == ch) {  
            strncpy (buf, pChar, MAX);  
            return buf;
```

```
        }  
        pChar++;  
    }
```

```
    return NULL;
```

```
}
```

```
int main () {  
    char *str = strsearch("Hallo Welt", 'W');  
  
    if (str != NULL)  
        printf ("Gefunden: %s\n", str);  
    return 0;  
}
```