Zeiger/Pointer in C

DI Thomas Helml

SEW

SJ 2015/16

Inhaltsverzeichnis

Pointer

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

Pointer

- Deklaration

```
Datentyp *name;
```

- `name` = Bezeichner
  - gleiche Namensregeln wie bei Variablen
- Typ der Variable ist `Datentyp *`
  - z.B. `int *name;`
  - `// Datentyp: int *`

Pointer

- Beispiel:

```
int *p;
```

  - Datentyp: `int *`
  - d.h. in `p` kann die Adresse eines `int`-Wertes gespeichert werden
- Adressoperator `&`
  - Liefert Adresse einer Variablen
  - `&p` => Adresse von `p`

Pointer

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

Pointer

- 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 *

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

Pointer

- 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
```

- Die Zuweisung `y=x` würde dasselbe machen
- `*ptr` ist wie eine Variable zu verwenden

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

Pointer

- Beispiel

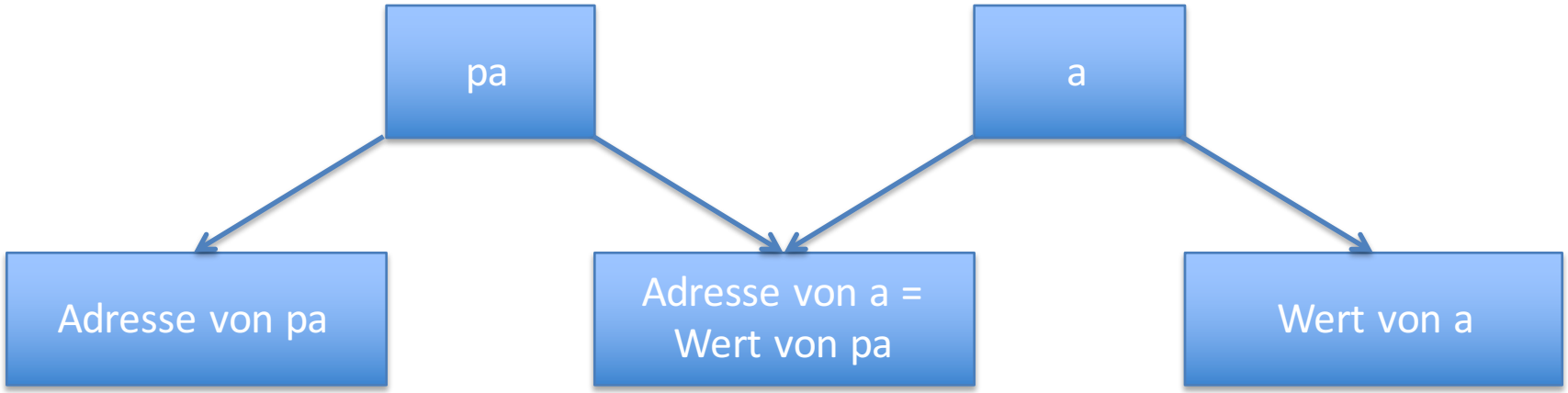
```
int a, b, *pa;

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;
```



&pa

&a

pa

a

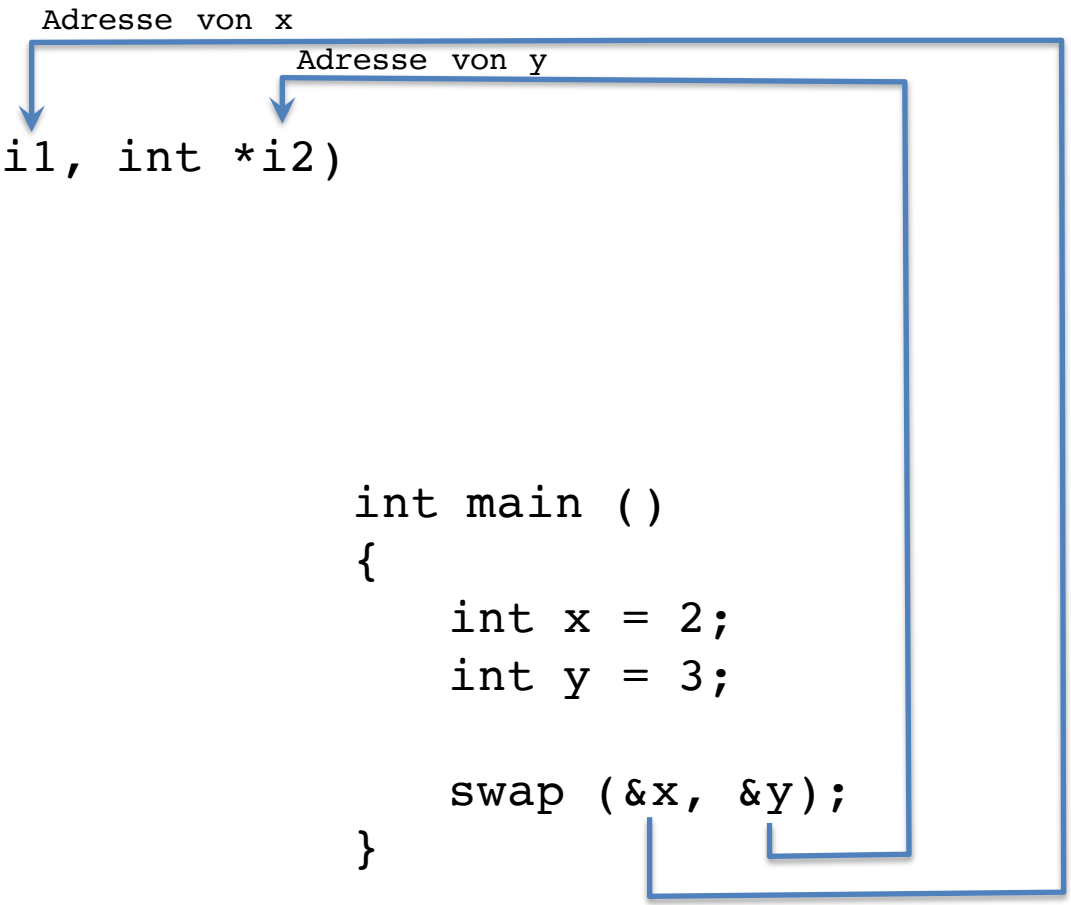
*pa

Call-by-Reference

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

Call-by-Reference

- 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

- Verweisoperator darf bei Zeiger mit gültiger Adresse verwendet werden
  - ansonsten: Segmentation fault! (Speicherzugriffsverletzung)
- DAHER:
  - Zeiger mit `NULL` initialisieren
  - 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

- 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

- Gegeben: Array a

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

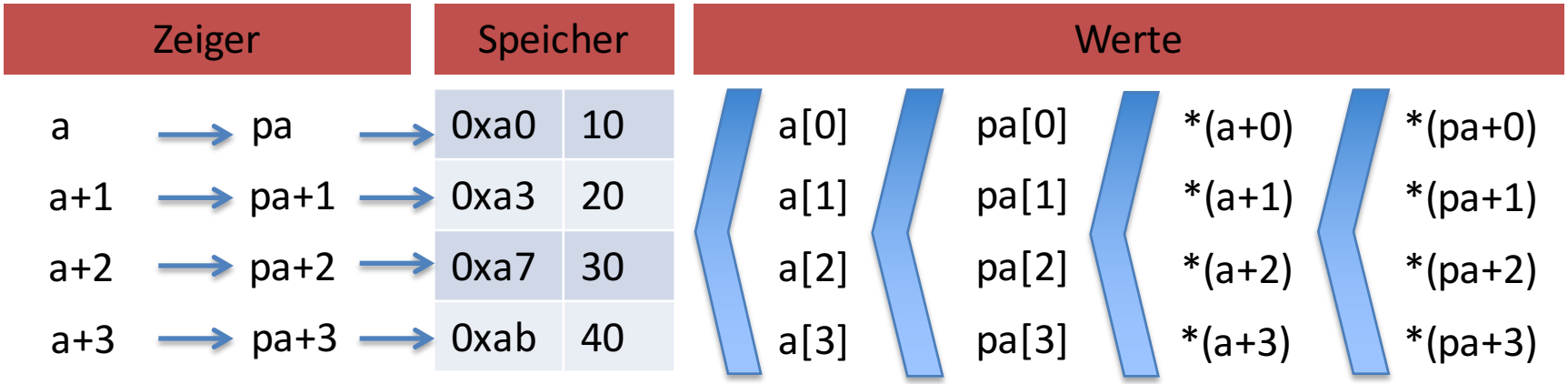
- In C:
  - a ist konstanter Zeiger auf 1. Array-Element a[0]
  - Somit kann a einem Zeiger zugewiesen werden

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

  - pa zeigt auf a[0]
  - pa+1 zeigt auf a[1]

Zeiger undArrays

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



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

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

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

pa = a;
```

- `pa + i` zeigt auf `a[i]`
- `pa = pa + i;`
  - Zeiger `pa` wird „versetzt“

Zeigerarithmetik

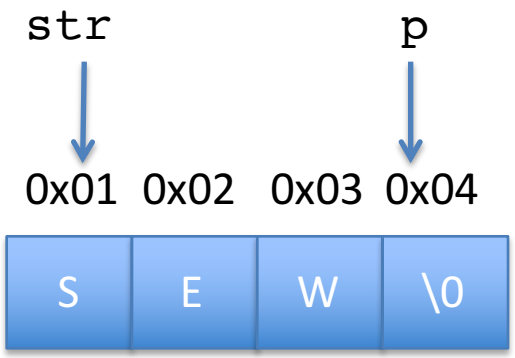
- Operator ++ und -- sind erlaubt, ABER
  - erhöhen immer Zeiger und nicht Inhalt!
  - `*pv++` entspricht `*(pv++)`
- Addition zweier Zeiger ist erlaubt (sinnlos)
- Subtraktion liefert Anzahl der Array Elemente zwischen Zeigern

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

Zeigerarithmetik

- 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);

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


    return 0;
}
```

AUSGABE?

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]);

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



    return 0;
}
```

AUSGABE?

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]);

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


    return 0;
}
```

AUSGABE?

Zeiger als Rückgabewert

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

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

- Verwendung primär bei
  - Arrays
  - Strings
  - Strukturen
- 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;
}
```