

Zeiger I

Lernziele

- Sie können Zeiger definieren und über den Zeiger die Variable, auf die er verweist, auslesen oder ändern.
- Sie können einen Zeiger als Parameter definieren und einer Funktion übergeben.
- Sie verstehen das Prinzip der Parameterübergabe Call-By-Reference.

Was ist ein Zeiger?

Zeiger im Alltag



Adresse eines
Hausbewohners

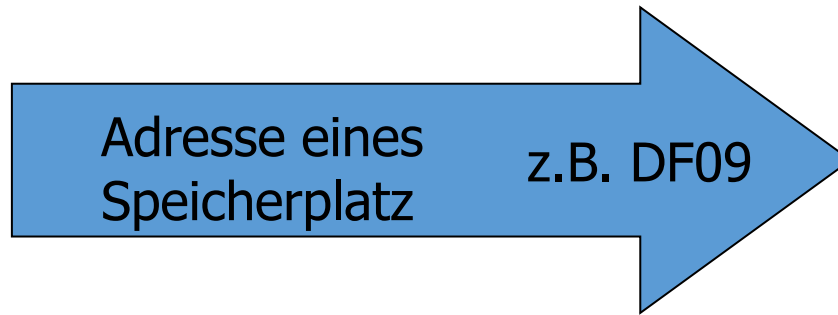


Zeiger in C

Adresse eines
Speicherplatz

z.B. DF09

Inhalt des Speichers



Wozu Zeiger?

- Um beim Funktionsaufruf die Parameter zu verändern.
- Um das Kopieren einer grossen Datenmenge beim Funktionsaufruf zu vermeiden.

```
#define dim 256  
void bluring(int a[][dim]);
```

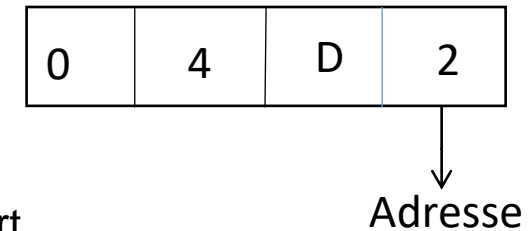
```
int main()  
{  
    int picture[dim][dim] = { 1, 2, 3, 4, 5 };  
    bluring(picture);  
}
```

Die Anfangsadresse des Arrays wird übergeben.

```
void bluring(int a[dim][dim]) {  
    for (int x = 0; x < dim; x++) { //for each row  
        for (int y = 1; y < dim; y++) { //for each column  
            a[x][y] = (a[x][y] + a[x][y - 1]) / 2;  
        }  
    }  
}
```

Variable und Adresse

- Eine Variable hat eine Adresse im Arbeitsspeicher und belegt eine bestimmte Anzahl Bytes um einen Wert zu speichern.
- Die Adresse einer Variablen kann durch Vorstellen des Adressoperators (&) herausgelesen werden.



```
int j = 1234;    //Zur Laufzeit:  
                // 1. Ein Speicherplatz von 4 Bytes wird an einer Adresse reserviert.  
                // 2. Der Wert 1234 (in Hex: 4D2) wird dorthin gespeichert.  
                // 3. Die Zahl wird beginnend vom tiefstwertigsten zum höchstwertigsten Byte abgelegt.
```

```
printf("%p", &j); //&j ergibt die Adresse der Variable j
```

Zeiger: Definition

- Definition:
 - sind Variablen
 - speichert die Adresse einer anderen Variablen

- Syntax:

```
Datentyp *zeigervariable;
```

Beispiel: `int *ptr;`

ptr ist ein Zeiger, welcher die Adresse einer Integer-Variablen aufnimmt.

Zeiger: Definition

Beispielprogramm

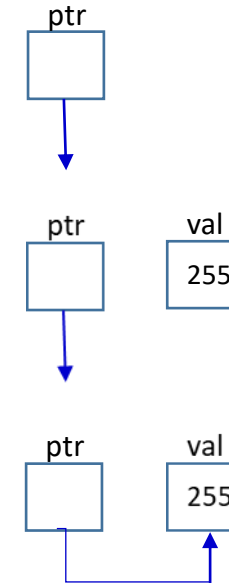
```
1. int *ptr;           //Definition
```

```
2. int val = 255;
```

```
3. ptr = &val;         //Zuweisung
```

```
printf("Adresse val: %p\n", &val);
```

```
printf("ptr zeigt auf die Adresse : %p\n", ptr);
```



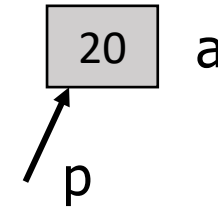
Zugriff auf die Zielvariable

Über einen Zeiger kann man auf die Zielvariable zugreifen (Dereferenzierung). Sie können damit über den Zeiger die Variable, auf die er verweist, auslesen oder ändern.

- Syntax: `*zeigervariable`

Inhaltsoperator

```
int a = 20, b;  
int *p = &a; // p zeigt nun auf a. (p erhält die Adresse von a)  
  
b = *p;      // *p ergibt den Wert der Variable a.  
  
*p = 5;      // 5 wird in den Speicherplatz geschrieben
```



*p ergibt den Wert von a

Inhaltsoperator *:

Hat man eine Adresse zur Verfügung, kann man mithilfe des Inhaltsoperators `*` auf den Wert, der an dieser Adresse gespeichert ist, zugreifen. Der Stern in der Definition eines Zeigers `*` ist nicht der Inhaltsoperator `*` zu verwechseln.

Zugriff auf die Zielvariable

```
int val = 255;
```

```
int *ptr = &val;    //Initialisierung
```

```
// Wert ausgeben
```

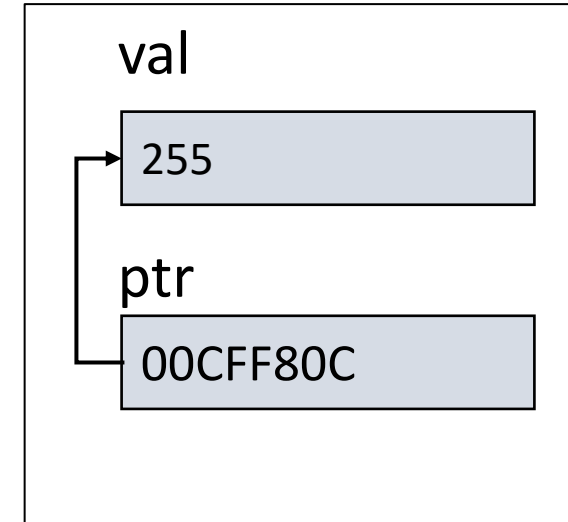
```
printf("*ptr : %d val : %d \n", *ptr, val);
```

```
// val neuen Wert zuweisen
```

```
*ptr = 128;    //äquivalent zu val=128
```

```
// Wert ausgeben
```

```
printf("*ptr : %d val : %d \n", *ptr, val);
```



Dereferenzierung.cpp

Initialisierung

- Zeiger müssen initialisiert werden.
- Verwendung von nicht initialisierten Zeiger
 - > unvorhersehbares Programmverhalten

```
int *p;  
int b = *p;
```

 - > Kompilierfehler
 - > Absturz: Irgendwo zufällige Variablen werden verändert.
- Best practice:
 - ✓ Zeiger werden immer initialisiert.
 - ✓ Noch nicht gebrauchte Zeiger soll man mit NULL initialisieren:

```
int *ptr = NULL;
```

NULL: eine vordefinierte Konstante (stdio.h)
 - ✓ Vor der Nutzung immer testen: `if (zeiger == NULL)`

Zeiger als Funktionsparameter

Zeiger können als Parameter definiert werden. In diesem Fall wird beim Aufruf eine Adresse übergeben.

```
void Modify(int *p, int c)
{
    c += 2;
    *p = *p + c; /* Veraendert Wert, auf den p zeigt */
}

int main()
{
    int x = 3, y = 7;

    printf("Vor dem Aufruf: x=%d, y=%d\n", x, y);
    Modify(&x, y); /* x wird veraendert, y nicht */
    printf("Nach dem Aufruf: x=%d, y=%d\n", x, y);

    return 0;
}
```

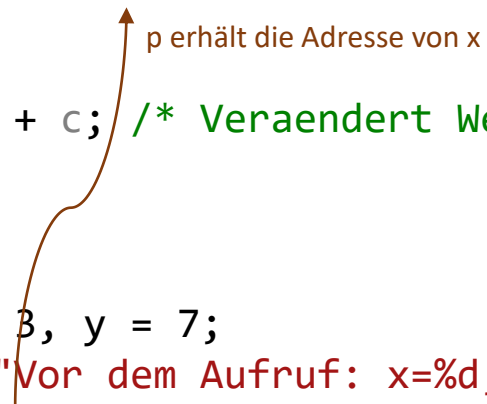
CallRef.cpp

Parameterübergabe: Call By Reference

- Einen Zeiger als Parameter definieren
- Die Adresse einer Variable als Argument an die Funktion übergeben

```
void Modify(int* p, int c)
{
    c += 2;
    *p = *p + c; /* Veraendert Wert, auf den p zeigt */
}
int main()
{
    int x = 3, y = 7;
    printf("Vor dem Aufruf: x=%d, y=%d\n", x, y);
    Modify(&x, y); /* x wird veraendert, y nicht */
    printf("Nach dem Aufruf: x=%d, y=%d\n", x, y);

    return 0;
}
```



p erhält die Adresse von x

CallRef.cpp

Werden als Funktionsübergabewerte Adressen (Zeiger) anstelle der Werte von Variablen übergeben, so können die Variablen nicht nur gelesen, sondern auch verändert werden. Diese Art der Parameterübergabe wird als "Call By Reference" bezeichnet.