Konzepte des prozeduralen Programmierens

Stand September 2022

- Prof. Dr. Oliver S. Lazar / Christian Frank

5 Zeiger

Zeiger (Pointer)

- Wer C beherrschen will, muss mit Zeigern umgehen können!
- Zeiger sind normale Variablen, die statt einem Wert eine Speicheradresse enthalten

Video
www.nerdwest.de

Was hat das für Vorteile?

- Mit Zeigern kannst du Datenobjekte direkt (call-by-reference) an Funktionen übergeben.
  - es muss also nicht jedes Mal die komplette Datenmenge übergeben werden, sondern nur die Adresse der Datenmenge
- z.T. einfacherer Umgang mit Arrays (Zeigerarithmetik → siehe Kapitel 6)
- Rekursive Datenstrukturen wie Listen und Bäume lassen sich fast nur mit Zeigern erstellen.

Beispiel

- Ein Zeiger sollte vom gleichen Datentyp sein, wie die Variable, auf die er zeigt.
- Ein Zeiger wird deklariert mit einem Sternchen (*) vor dem Namen.
- Zuletzt muss die korrekte Adresse der Variablen dem Zeiger zugewiesen werden.
  - diese erhalten wir mit dem kaufmännischen-Und &

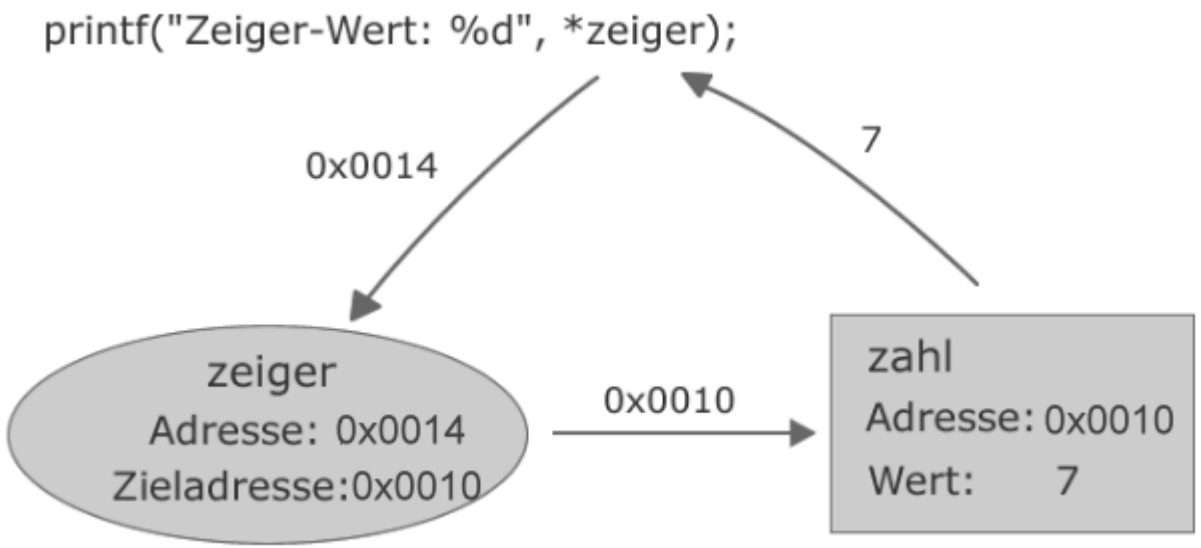
```
int zahl = 7;
int *zeiger;
// int *zeiger = &zahl;
zeiger = &zahl;

printf("Zeiger-Wert: %d\n", *zeiger);
```

```
Zeiger-Wert: 7
```

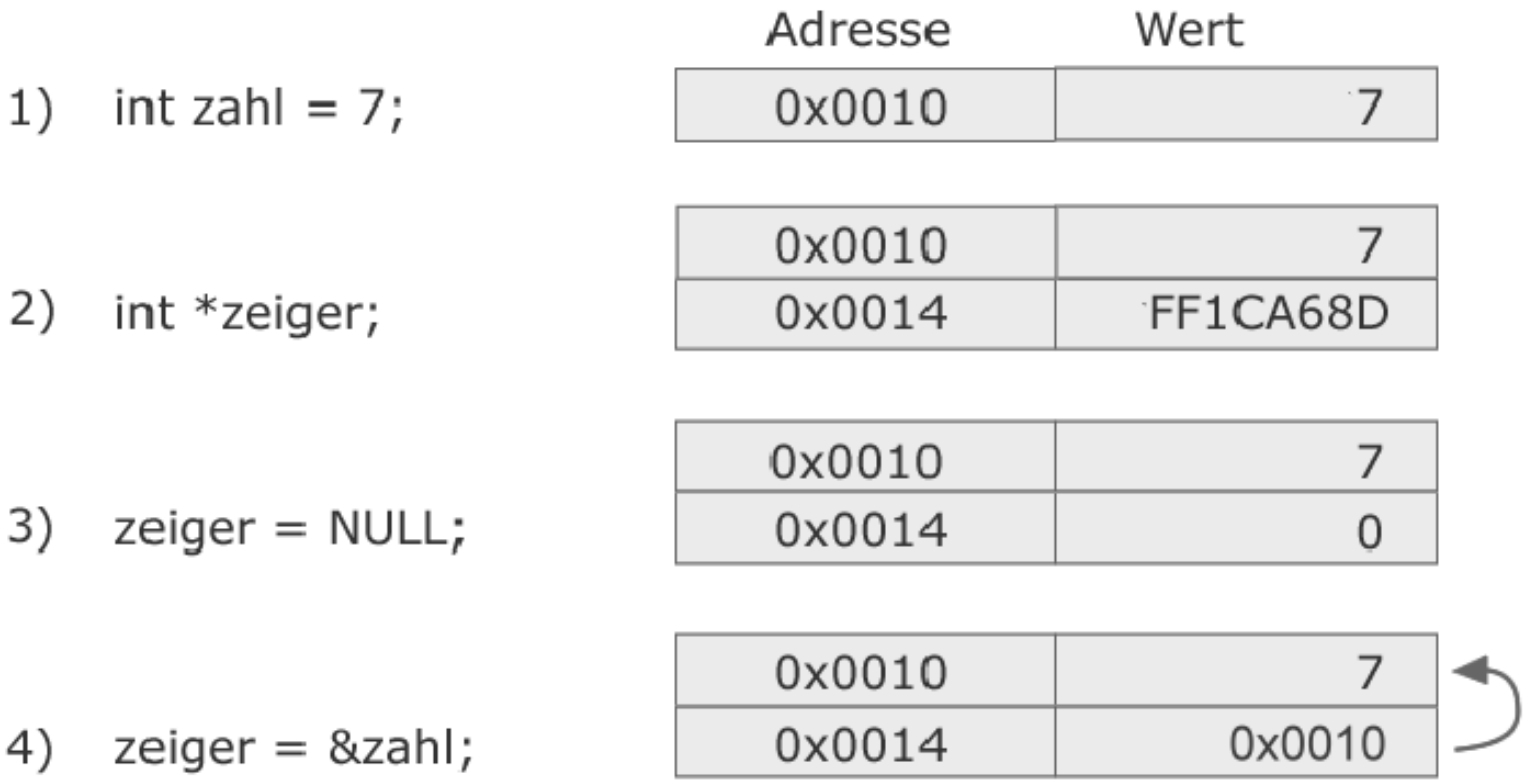
Beispiel - Fortsetzung

- Ein Zeiger repräsentiert eine Adresse und nicht wie eine Variable einen Wert.
- Möchte man auf den Wert der Adresse zugreifen, auf die ein Zeiger zeigt, muss der Stern * vor den Namen gesetzt werden.



Beispiel - Fortsetzung

- So sieht es dann im Speicher aus



Nullzeiger

- Um zu vermeiden, dass ein nicht gesetzter Zeiger im Programm verwendet wird, kann man aus diesem einen sogenannten **Nullzeiger** machen.
- Damit lässt sich vor einem Zugriff auf den Zeiger dessen Verwendbarkeit prüfen.

```
int zahl = 7;
int *zeiger = NULL;

if(zeiger != NULL) {
    printf("Versuch 1, Zeiger-Wert: %d\n", *zeiger);
}

zeiger = &zahl;

if(zeiger != NULL) {
    printf("Versuch 2, Zeiger-Wert: %d\n", *zeiger);
}
```

```
Versuch 2, Zeiger-Wert: 7
```

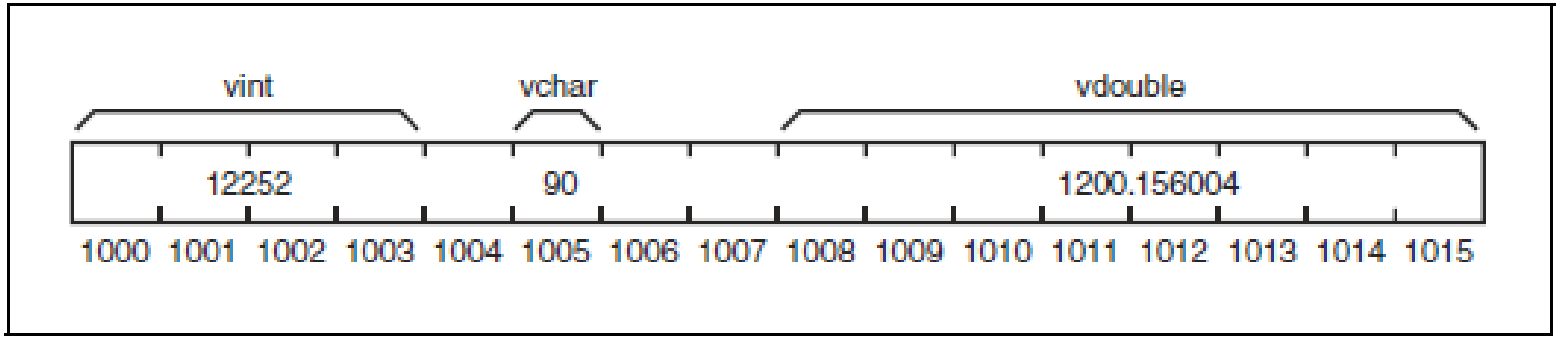
Zeiger und Variablentypen

- Der Speicherbedarf der verschiedenen Variablentypen ist unterschiedlich.
  - ein `char` belegt i.d.R. 1 Byte, ein `int` 4 Bytes, ein `double` 8 Bytes u.s.w.

Wie gehen Zeiger mit Adressen von Variablen um, die mehrere Bytes belegen?

- Die Adresse einer Variablen bezeichnet immer das erste (niedrigste) Byte, das die Variable im Speicher belegt.

```
int vint = 12252;
char vchar = 90;
double vdouble = 1200.156004
```



Zeiger auf Zeiger

- Zeiger können auch auf Adressen von anderen Zeigern zeigen
  - Dies erreicht man mit dem doppelten Stern-Operator **

```
int zahl=7;
int *zeiger = &zahl;
int **zeigerAufZeiger = &zeiger;

printf("Wert von zeigerAufZeiger -> zeiger -> zahl: %d\n", **zeigerAufZeiger);
```

```
Wert von zeigerAufZeiger -> zeiger -> zahl: 7
```

Aufgabe 05.02

Zeiger und Variablentypen

Adresse	Inhalt	Name	Typ
1000	9	b	int
1001			
1002			
1003			
1004	7	a	int
1005			
1006			
1007			
1008	1004	zeiger1	int*
1009			
1010			
1011			
1012	1008	zeiger2	int**
1013			
1014			
1015			

Frage	Antwort
• &b • &zeiger1 • &zeiger2 • *zeiger1 • *zeiger2 • **zeiger2 • &a	

Klausur WS-15/ Aufg. 10

Aufgabe 05.01

Berechne von Hand in jeder Zeile die Werte der Variablen und Zeiger, die sich verändern.

```
int a=2, b=5, *c=&a, *d=&b;
```

Anweisung	Wert a	Wert b	Wert *c	Wert *d
`a = *c * *d;`				
`*d -= 3;`				
`b = a * b;`				
`c = d;`				
`b = 7;`				
`a = *c + *d;`				

Klausur WS-15-Nachschreib/ Aufg. 10

Aufgabe 05.03

Deklariere eine integer-Variable und weise ihr den Wert 5 zu. Gib den Wert 5 einmal direkt (Nutzung der Variablen selbst) und einmal indirekt (über eine Zeigervariable) aus.

Zu verwendende Codeelemente:

- `int i;`
- `int *i_zeiger;`
- `printf("%d",i);`
- `printf("%d",*i_zeiger);`

Erweiterung

- Lass nun auch die Adresse deiner integer-Variablen über zwei Wege ausgeben.

Zeiger als Funktionsparameter

- *call-by-value* haben wir bereits kennengelernt (Parameterübergabe und Rückgabewert per `return`)
  - Nachteil: bei jedem Aufruf müssen alle Parameter kopiert werden
- schöner und schneller geht das mit *call-by-reference*
  - statt Variablen werden Speicheradressen übergeben
- Beispiel siehe nächste Folie

Beispiel - Addieren

```
#include<stdio.h>

void addiere(int *summand1, int *summand2, int *summe) {
    *summe = *summand1 + *summand2;
}

int main() {
    int zahl1 = 6;
    int zahl2 = 3;
    int summe;

    addiere(&zahl1, &zahl2, &summe);

    printf("Summe von zahl1 und zahl2: %d\n", summe);
    return 0;
}
```

```
Summe von zahl1 und zahl2: 9
```

Aufgabe 05.04

Schreibe eine Funktion `void einlesen (int*, int*)`, die zwei ganzzahlige Werte einliest. Rufe diese Funktion mit zwei in der main-Funktion deklarierten int-Variablen (bzw. mit deren Adressen) auf. Gib nach dem Einlesen das Produkt der beiden Zahlen auf dem Bildschirm aus

Zu verwendende Codeelemente:

- `void einlesen(int* z1, int* z2);  // Funktionsdeklaration`
  - lesen Sie die Werte für die Variablen über `scanf` innerhalb der Funktion `einlesen` (s.u.) ein
- `einlesen(&zahl1,&zahl2);  // Funktionsaufruf`

Aufgabe 05.05

Lies den Radius eines Kreises ein. Mit Hilfe zweier Funktionen soll der Umfang und der Flächeninhalt berechnet werden.

a) Übergabe der Eingabeparameter über Zeiger; Rückgabe des Ergebnisses per `return`

b) Übergabe der Eingabeparameter und des Ergebniswertes über Zeiger

Zu verwendende Codeelemente:

zu a)
```
float berechneUmfang(float* r);      // Funktionsdeklaration
float berechneFlaeche(float* r);     // Funktionsdeklaration
```

zu b)
```
void berechneUmfang(float* r,float* erg);  // Funktionsdeklaration
void berechneFlaeche(float* r,float* erg); // Funktionsdeklaration
```

Klausur WS 16-17/ Aufg.8