Zeigerarithmetik

Zeigerarithmetik = „Rechnen mit Adressen“

- erlaubt: Addition, Subtraktion und Vergleich von Zeigern
- wichtig: Ergebnis hängt von Typ des Zeigers ab
- nicht erlaubt bei Zeigern auf Funktionen
- nicht erlaubt: Multiplikation, Division von Zeigern etc.

Beispiele:

```
int *zeiger1, *zeiger2;

zeiger1 = zeiger1 + 4;          ⇒   Addition

if (zeiger1  = =  zeiger2)      ⇒   Vergleich
   {
    ...
   }
```

Beispiel: Addition bei Zeiger vom Typ “char”

```
char zeichen[] = "Text";
char *zeiger;
```

```
zeiger = zeichen;
```

Zeigerinhalt nach der Zuweisung: 800

```
zeiger = zeiger + 2;
```

Zeigerinhalt nach der Zuweisung: 802

Adresse	Inhalt	Name
800		zeichen[0]
801		zeichen[1]
802		zeichen[2]
803		zeichen[3]
804		zeichen[4]
805		
806		
807		
808		
809		
810		
811		

`printf("%c",*zeiger);` → ergibt **x** auf dem Bildschirm

→ Operation „zeiger + 2“ erhöht die Adresse um 2
Zeiger „zeigt“ auf das übernächste Zeichen

Beispiel: Addition bei Zeiger vom Typ “int”

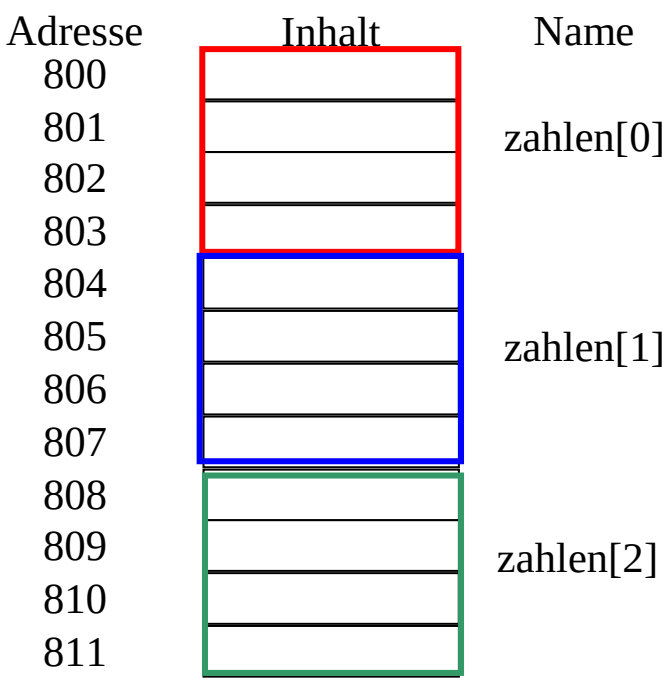
```
int zahlen[] = {7,2,9};
int *zeiger;
```

```
zeiger = zahlen;
```

Zeigerinhalt nach der Zuweisung: 800

```
zeiger = zeiger + 2;
```

Zeigerinhalt nach der Zuweisung: 808



`printf("%d",*zeiger);` → ergibt **9** auf dem Bildschirm

→ Operation „zeiger + 2“ erhöht die Adresse um 8
Zeiger „zeigt“ auf die übernächste Zahl

Additionsoperation mit Zeigern

Operation: `zeiger = zeiger + n;`

Bedeutung: Zeiger soll auf die n-te Variablen des entsprechenden Typs zeigen
Inhalt des Zeigers (= eine Adresse) ändert sich abhängig vom Typ
Adresserhöhung entspricht dem Speicherbedarf des Typs

Beispiel: zeiger++ für verschiedene Zeigertypen

Zeigertyp	Speicherbedarf	Zeigerinhalt
`char`	1 Byte	Adresse erhöht sich um 1
`short int`	2 Byte	Adresse erhöht sich um 2
`int`	4 Byte	Adresse erhöht sich um 4
`float`	4 Byte	Adresse erhöht sich um 4

usw...

Überprüfung der Zeigerarithmetik mit Hilfe von sizeof()

Anweisung **`sizeof ( )`** ermittelt Speicherbedarf einer Variablen oder eines Typs

z.B.: **`wert = sizeof(char);`** ergibt 1

bzw:
```
float zahl;
wert = sizeof(zahl);
```
ergibt 4

Überprüfung der Zeigerarithmetik:

```
int zahl, *zeiger;
zeiger = &zahl;
printf("%X",(int)zeiger);
printf("%X",(int)(zeiger+1));
printf("%d", sizeof(int));
```

Bildschirmausgabe

`FBB`	Adresse von **`zahl`**
`FBF`	Adresse nächste Variable
`4`	Differenz der Adressen

Speicherbedarf für Zeiger

Speicherbedarf ist abhängig von Betriebssystem und Speichermodell

Speichermodell legt fest, wie groß der Speicherbedarf des Programms werden darf

→ Speichermodell wird bei der Compiler-Konfiguration festgelegt.

→ Compiler wählt dann passende Größe für alle Zeiger

z.B.:	"**`near`**" Zeiger:	16 Bit	(Adressraum 64 kByte)
	"**`far`**" Zeiger:	32 Bit	
	usw...		

Speichermodell kann auch explizit bei der Deklaration eines Zeigers angegeben werden:

Beispiel: Deklaration eines "**`far`**" Zeigers:

```
char far *zeiger;
```

Zeiger auf Datenstrukturen

- auch für Datenstrukturen können Zeiger deklariert werden
- wichtig: Syntax des Zugriffs auf Komponenten ist anders

Definition einer Datenstruktur

```
struct person {
                char name[20];
                int alter;
              };
```

Deklaration einer Variablen

```
struct person Maier;
```

Deklaration eines Zeigers

```
struct person *zeiger;
```

Zugriff auf Komponenten einer Datenstruktur:

direkter Zugriff bei einer **`struct`** Variablen:

Auswahl von Komponenten über den Punkt-Operator.

```
variable.alter = 22;
```

indirekter Zugriff bei einem Zeiger auf eine **`struct`** Variable

Auswahl von Komponenten über den Pfeil-Operator.

```
zeiger->alter = 22;
```

Allgemeine Syntax

```
Struktur_Variablenname.Komponente
```

bzw.:

```
Struktur_Zeigername->Komponente
```

Beispiel: direkter Zugriff

```
struct person Maier;
Maier.alter = 22;
```

Beispiel: indirekter Zugriff

```
struct person Maier;
struct person *zeiger;

zeiger = &Maier;
zeiger->alter = 22;
unterprogramm(&Maier);
```

wichtig für Unterprogramme

```
void unterprogramm(struct person *px)
   {
      px->alter = 33;
   }
```

Übergabe von Strukturvariablen an Funktionen:

- direkt:

```
unterprogramm(struct person px);
```

→ „call by value“
Variable im Hauptprogramm bleibt unverändert

- indirekt über Zeiger:

```
unterprogramm(struct person *px);
```

→ „call by reference“
Inhalt der Variable im Hauptprogramm kann im Unterprogramm verändert werden

Einige Klausuraufgaben zum Thema „Zeiger“

Aufgabe x (Punktzahl: 6): Welche der folgenden Anweisungen a) bis g) sind korrekt und welche sind fehlerhaft? Begründen Sie jeweils kurz Ihr Ergebnis. **Begründen Sie jeweils kurz Ihr Ergebnis** (Ohne Begründung gibt es keine Punkte !!!). Falls Die Anweisung korrekt ist, geben Sie bitte an, welcher Wert der betroffenen Variablen zugewiesen wird. Die Variable `a` sei an der Adresse `500,` die erste Variable des Arrays `b` an der Adresse `1000` , die Variable `c` sei an der Adresse `2000,` der Zeiger `d` sei an der Adresse `5000` gespeichert.

```
char a, b[]="Text";
int c, *d;
```

a) `a = b;`

b) `a = b[4];`

c) `a = b[10];`

d) `a = &b;`

e) `b = (int)d;`

f) `c = (int)d;`

Aufgabe y (Punktzahl: 6): Geben Sie jeweils den Programmcode an, der die gestellte Aufgabe erfüllt. Benutzen Sie die hier deklarierten Variablen und Zeiger:

```
char x, *y, z[]="text";
```

a) Der Variablen **x** den Inhalt des 3. Elements des Arrays **z** zuweisen:

b) Der Variablen **x** den Inhalt der Speicherstelle zuweisen, auf die der Zeiger **y** zeigt:

c) Dem Zeiger **y** die Adresse von **x** zuweisen:

d) Dem Zeiger **y** die Adresse des 3. Elements des Arrays **z** zuweisen:

e) Dem 1. Element von z den Inhalt von x zuweist

f) Das String-Ende-Zeichen des Arrays z mit dem ASCII-Code von `'e'` überschreibt

Zusammenfassung: Zeiger

- Zeiger: enthalten Adressen (von Variablen)
  benötigen selbst auch Speicherplatz

- Adreß-Operator: `&` ergibt Adresse einer Variablen

- Dereferenzierung: `*` ermöglicht Zugriff auf Variable, auf die der Zeiger zeigt

- Unterprogramme: "call-by-value" = Daten werden übergeben

  "call-by-reference" = Zeiger auf Daten wird übergeben

- Arrays: Kurzschreibweise: Name ohne `[]` = Adresse des Arrays

- Zeiger können nicht einfach mit **`printf()`** ausgegeben werden → Typumwandlung

- Zugriff auf Komponenten einer Datenstruktur mittels Zeiger: `->` Pfeil Operator