



## 13. Programmiersprache C (Fortsetzung)

### 13.10 Inkrement/Dekrement Operator

"++" und "--" sind unäre Operatoren zur Inkrementierung (hochzählen) und Dekrementierung (runterzählen) von ganzzahligen Variablen.

<code>++i</code>	<i>Pre-inkrement:</i>	i erst um 1 erhöhen und dann verwenden.
<code>i++</code>	<i>Post-inkrement:</i>	i erst verwenden und dann um 1 erhöhen.
<code>--i</code>	<i>Pre-dekrement:</i>	i erst um 1 erniedrigen und dann verwenden.
<code>i--</code>	<i>Post-dekrement:</i>	i erst verwenden und dann um 1 erniedrigen.

#### Beispiel:

```
int i=0, k=0, m=0, n=0;
printf("%2d %2d %2d %2d \n" , ++i, k++, --m, n--);
/* druckt:  1  0 -1  0 */
printf("%2d %2d %2d %2d \n" , i, k, m, n);
/* druckt:  1  1 -1 -1 */
```



## 13.10 Inkrement/Dekrement Operator (Fortsetzung)

Die Position von "++" und "--" in der Vorrangtabelle ist noch vor den arithmetischen Operatoren "\*" und "/" .

[ ] ( ) . ->	Auswertung von links nach rechts
! ~ ++ -- - (type) & *	Auswertung von rechts nach links
* / %	Auswertung von links nach rechts
+ - (binär)	Auswertung von links nach rechts
>> <<	Auswertung von links nach rechts
< <= > >=	Auswertung von links nach rechts
== !=	Auswertung von links nach rechts
...	...
&&	Auswertung von links nach rechts
	Auswertung von links nach rechts
?:	Auswertung von rechts nach links
= += -= *= etc.	Auswertung von rechts nach links
,	Auswertung von links nach rechts



## 13.11 Bedingte Ausdrücke

Wenn in if-Ausdrücken abhängig vom Vergleichsergebnis ein Wert zugewiesen wird, kann stattdessen kürzer die "*bedingte Bewertung*" verwendet werden.

### Syntax:

*Wert* = *Vergleichsausdruck* '?' *ausdruck1* ':' *ausdruck2*

### **Beispiel:**

```
max = x > y ? x : y;  
/* ist gleichbedeutend mit */  
if (x > y) max = x;  
else     max = y;
```



## 13.11 Bedingte Ausdrücke (Fortsetzung)

Die Position des "bedingten Ausdrucks" in der Vorrangtabelle ist sehr niedrig kurz vor der Zuweisung .

[ ] ( ) . ->	Auswertung von links nach rechts
! ~ ++ -- - (type) & *	Auswertung von rechts nach links
* / %	Auswertung von links nach rechts
+ - (binär)	Auswertung von links nach rechts
>> <<	Auswertung von links nach rechts
< <= > >=	Auswertung von links nach rechts
== !=	Auswertung von links nach rechts
...	...
&&	Auswertung von links nach rechts
	Auswertung von links nach rechts
?:	Auswertung von rechts nach links
= += -= *= etc.	Auswertung von rechts nach links
,	Auswertung von links nach rechts



## 13.12 Spezielle Zuweisungen

Häufig werden Ausdrücke der Art

$$\text{Var} = \text{Var} + 1$$

formuliert, d.h. Var ist auf beiden Seiten der Zuweisung.

Diese Ausdrücke lassen sich in C durch spezielle Zuweisungsoperatoren effizienter schreiben.

Var += 5	statt	Var = Var + 5
Var -= 5	statt	Var = Var - 5
Var *= 5	statt	Var = Var * 5
Var /= 5	statt	Var = Var / 5
Var %= 5	statt	Var = Var % 5



## ÜBUNG: Spezielle Operatoren

Was wird durch folgendes Programm ausgedruckt?  
(C-Puzzle, kein ernsthafter Code)

```
int x=1, y=1, z=1;

/*--- 1 ---*/
x+=++y;
printf("x:%d    y:%d\n", x, y);

/*--- 2 ---*/
printf("z:%d\n", z+=x<y?y++:x++);
printf("x:%d    y:%d    z:%d \n", x, y, z);

/*--- 3 ---*/
x=z=1;
while(++z<4){
    printf("%d %d\n", x++, x++);
}
```



## Fazit:

Zur Vermeidung sehr schwer verständlicher und debugbarer Ausdrücke gilt:

- Increment/Decrement-Operatoren
- bedingte Ausdrücke und
- spezielle Zuweisungen

sollten nur sehr behutsam (verantwortungsbewußt) eingesetzt werden !



## 13.13 Adressen und Zeiger

### Zeiger

- eine Variable, die eine Adresse enthält
- ist Typ-gebunden

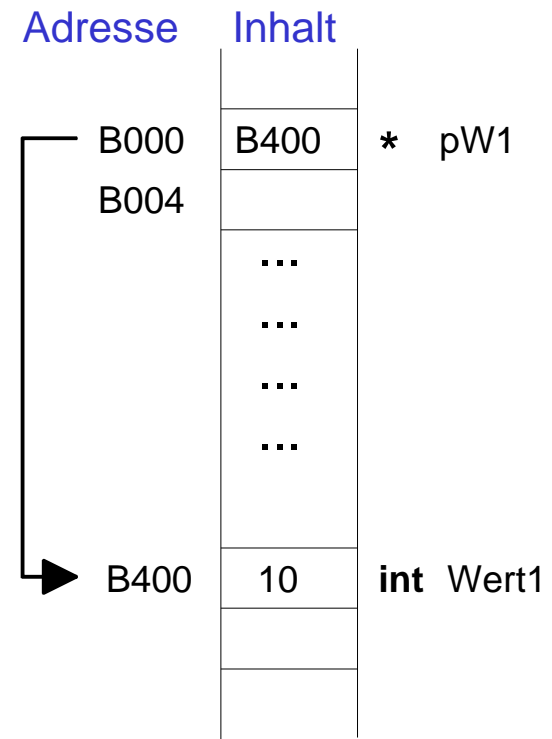
### Deklaration:

- durch **\*** vor Variablenname

### Initialisierung:

- Zeiger muß mit gültiger Adresse belegt werden
- Die Adresse einer Variablen erhält man durch den Adressoperator &.

```
/* Deklaration */  
int  Wert1 = 10;  
int  *pW1;  // pW1 ist Zeiger auf Integer  
  
/* Initialisierung */  
pW1 = &Wert1;
```







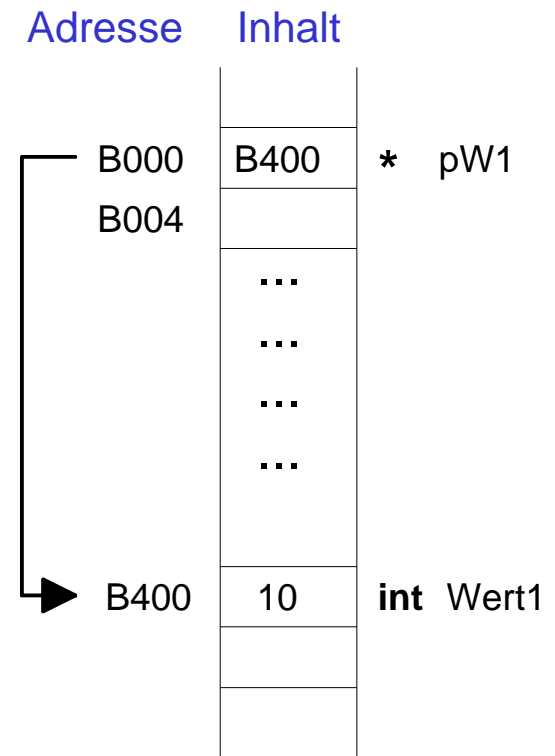
## 13.14 Zugriff auf Daten über den Zeiger

**Zugriff auf den Zeigerwert** (= Wert von derjenigen Variablen holen, auf die der Zeiger zeigt)  
 ---> durch `*` -Operator vor dem Zeigernamen

```
/* Deklaration */
int Wert1 = 10, *pW1;

/* Initialisierung */
pW1 = &Wert1;

/* Zugriff */
printf("Wert %d, Adresse %X", *pW1, pW1);
```



### WICHTIG:

In der Deklaration bedeutet `*pWert1`: "**pWert1 ist ein Zeiger auf Typ .....**"  
 Im Programmtext bedeutet `*pWert1`: "**Wert, auf den der Zeiger pWert1 zeigt**"  
 (→ Dereferenzierung)



## 13.14 Zugriff auf Daten über den Zeiger (Fortsetzung)

### Achtung: Fehlergefahr

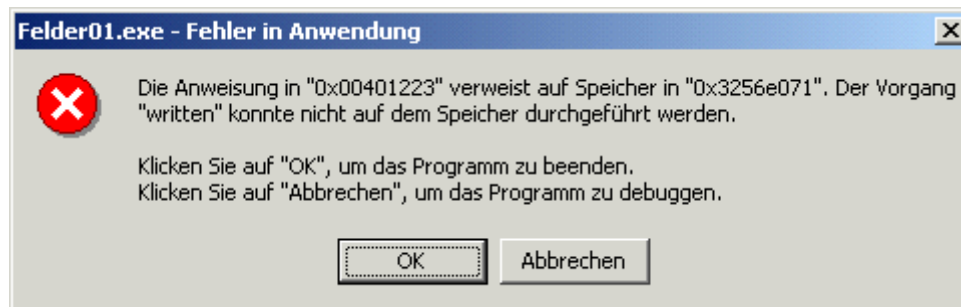
--> "dangling pointer"

```
/* Deklaration */
int *pW1;

/* Zugriff */
*pW1=10; // Der Wert auf den der
          // Zeiger zeigt soll 10 sein.
```

..... aber wohin zeigt der Zeiger ?

Adresse	Inhalt	
B000	????	* pW1
B004		
	...	
	...	
	...	
	...	





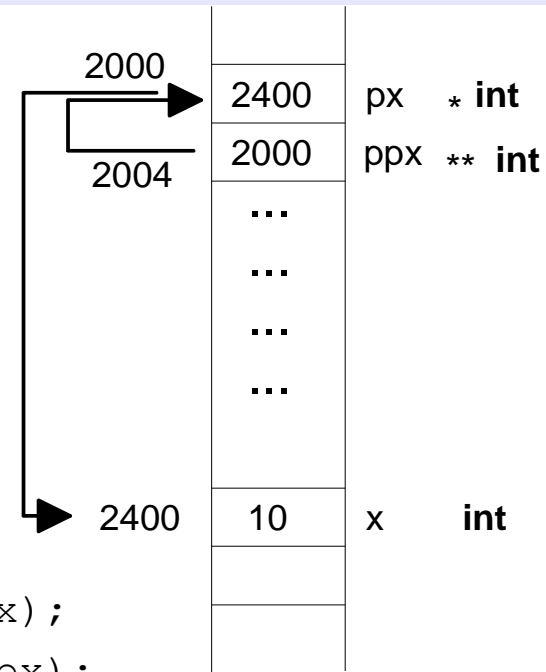
## ÜBUNG: Einfache Zeigeroperationen

Gegeben ist folgendes Programm:

```
int x = 10;
int *px;
int **ppx;
```

```
px = &x;
ppx = &px;
```

```
printf("%d %d %d", x, &x, px);
printf("%d %d %d", &px, *px, *ppx);
printf("%d %d %d", &ppx, **ppx, ppx);
```



Gegeben sei nebenstehendes Speicherbild (Memorymap).  
Was wird ausgedruckt ?



## 13.15 Ein- und mehrdimensionale Felder

### 13.15.1 Definition

**Feld:** Menge von gleichartigen Variablen, die durch einen einzelnen Namen repräsentiert werden

**Definition:** *Typ Variablenname [ Anzahl der Elemente ]*

**Zugriff:** Index des ersten Feldes ist 0 !

```
/* Definition */
int x[2];

/* Initialisierung */
x[0]=12;
x[1]=25;

printf('x0 ist %d \n', x[0]);
```

```
/* Definition */
char txt[2];

/* Initialisierung */
txt[0]='A';
txt[1]='b';

printf('x0 ist %c \n', txt[0]);
```

**Fazit:** Es gibt keinen Unterschied zwischen Feldern mit Zahlen und Zeichen !  
Zeichen werden auch als Zahlen (ASCII) abgespeichert.



## 13.15.2 Definition und Initialisierung

**/\* mit Angabe der Feldgrösse \*/**

```
int x[10] = {3, 6, 9, 12, 15, 18, 21, 24, 27, 30};
```

```
char txt[5] = {'a', 'b', 'c', 'd', 'e' };
```

**/\* ohne Angabe der Feldgrösse \*/**

```
int x[] = {3, 6, 9};
```

```
char txt[] = {'a', 'b', 'c', 'd', 'e' };
```

**/\* Teilinitialisierung \*/**

```
int x[10] = {3, 6};
```

```
char txt[5] = {'a', 'b'};
```



### 13.15.3 Mehrdimensionale Felder

```
/* mit Angabe der Feldgrösse */
int x2[2][3];

char c2[3][2] = {{ 't', 'f' }, /* Feld von 3 Elementen */
                 { 'f', 'f' }, /* mit jew. 2 Elementen */
                 { 'f', 'x' } };

char c2[][2] = {{ 't', 'f' }, /* Felddimension 1 */
                { 'f', 'f' }, /* kann entfallen. */
                { 'f', 'x' } };

int x3[4][3][2] = { { { 0,1 }, { 1,0 }, { 1,1 } },
                    { { 1,1 }, { 0,0 }, { 1,0 } },
                    { { 0,0 }, { 1,1 }, { 1,1 } },
                    { { 0,1 }, { 0,1 }, { 0,1 } } };
```

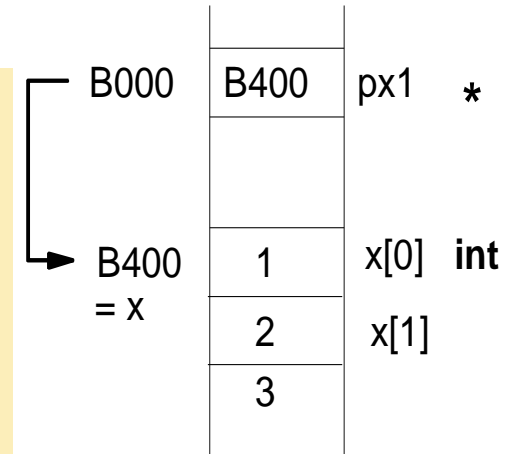
**Achtung:** Die Klammern werden vom Compiler nicht ausgewertet !



## 13.16 Felder und Zeiger

```
int x[] = {1, 2, 3, 4, 5, 6, 7};
int *px1 = x;          /* = &x[0] */

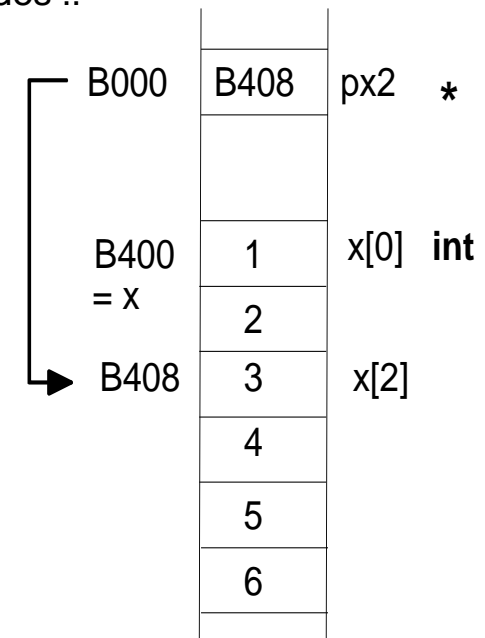
/* Folgende Schreibweisen sind äquivalent */
printf("Adr. des 1. Zeichens=%X", &x[0]);
printf("Adr. des 1. Zeichens=%X", x);
printf("Adr. des 1. Zeichens=%X", px1);
```



**Wichtig:** Der Feldname ohne Index liefert die Adresse des 1. Feldes !!

```
int x[] = {1, 2, 3, 4, 5, 6, 7};
int *px2 = &x[2];

/* Ausgabe : 3 3 6 */
printf("%d %d %d", *px2, px2[0], px2[3]);
```



**Wichtig:** Ein indizierter Zeiger liefert den indizierten Wert !!



## 13.16 Felder und Zeiger (Fortsetzung)

```
/* Definition und Initialisierung */
char x[4][3][2] =
    { { { 0, 1}, { 2, 3}, { 4, 5} },
      { { 6, 7}, { 8, 9}, {10,11} },
      { {12,13}, {14,15}, {16,17} },
      { {18,19}, {20,21}, {22,23} } };

/* Zugriff auf Feldelemente */
a=3; b=2; c=1;
y = x[a][b][c];      /* y= 23*/
```

Adresse	Inhalt
$x = x[0] = x[0][0] = 1000$	00
	01
1002	02
	03
1004	04
	05 $x[0][2][1]$
$x[1] = x[1][0] = 1006$	06
	07
1008	08
	09
$x[1][2] = 100A$	10 $x[1][2][0]$
	11
	12
	13
	14
	15
	16
	17
	18
	19
	20
	21
	22
	23

**Wichtig:** Unvollständige Feldnamen (fehlende Klammern) sind Adressen und können wie Zeiger verwendet werden.

### Beispiele:

- $x = x[0] = x[0][0] = 1000$
- $x[1] = x[1][0] = 1006$
- $x[1][2] = 100A$
- $x[1][2][0] = 10$  (der Wert !!)





## ÜBUNG: Umgang mit Zeigern

Zeichnen Sie die Memorymap. Was wird ausgedruckt ?

```
int a[]={1,2,3,4,5,6,7};

int main() {
    int *p1;
    int **p2;

    p1=&a[2];
    p2=&p1;

    printf("%d\n",a[3]);
    printf("%d\n",*p1);
    printf("%d\n",p1);
    printf("%d\n",&p1);
    printf("%d\n",p2);
    printf("%d\n",&p2);
    printf("%d\n",p2[0][2]);

    return 0;
}
```

### Folgende Annahmen gelten:

Das Feld a beginne bei Adresse 1000 (dezimal).

Der Zeiger p1 stehe bei Adresse 2000 (dez.)  
gefolgt vom Zeiger p2.

### Achtung:

Das Beispiel hat didaktischen Wert, zeugt aber nicht gerade von gutem Stil !



## 13.17 Zeichenketten (Strings)

### 13.17.1 Definition und Initialisierung

#### String

- Sequenz von Zeichen,
- in Anführungszeichen eingeschlossen,
- mit '\0' abgeschlossen ist (Null-Zeichen).

```
char txt1[] = "AB1";  
char txt2[4] = "AB1"; /* Platz für Nullzeichen nicht vergessen */  
char txt3[] = {'A', 'B', '1', '\0'};  
  
/* String ausgeben mit %s und Stringname */  
printf("%s %s %s", txt1, txt2, txt3);
```



= 0x41 0x42 0x31 0x0

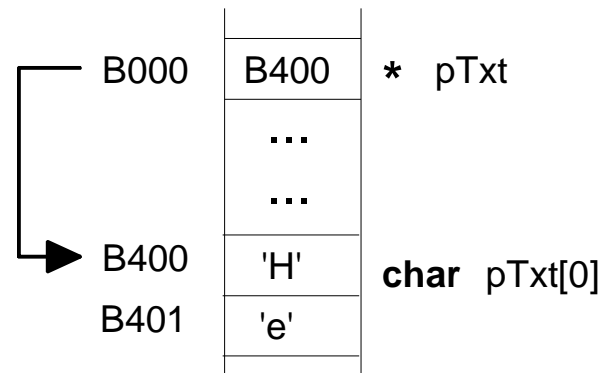
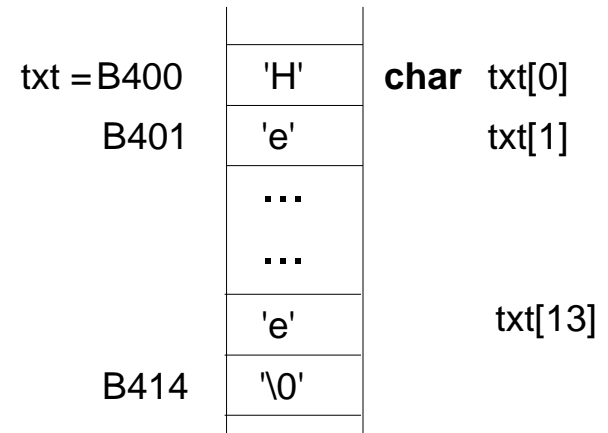


## 13.18 Strings und Zeiger

Feldnamen können wie Zeiger verwendet werden,  
(sind aber nur Adressen) !

```
char txt[] = "Herr der Ringe";
printf("Adr. des 1. Zeichens=%X", txt);
```

```
char *pTxt = "Herr der Ringe";
printf("Adr. des 1. Zeichens=%X", pTxt);
```





## ÜBUNG: Beispiele zu Feldern und Strings

1. Schreiben Sie ein C-Programm, welches eine 4x3-Matrix mit einem Vektor multipliziert.
2. Schreiben Sie ein C-Programm, welches die Grossbuchstaben eines Strings zählt und ausgibt.



## 13.19 Funktionen mit Call-by-reference-Parameterübergabe

```
/* Funktionsdeklaration */
void SetString2x(char *str);

/* Aufrufendes Programm */
int main () {

    char txt[] = "Autobahn";

    /* Funktionsaufruf */
    SetString2x(txt);
    printf("%s", txt);

    return 0;
}
```

```
void SetString2x(char *str){

    int i=0;

    while(str[i] != '\0'){
        str[i]='x';
        i++;
    }
}
```

**Wichtig:** Mit Hilfe von Zeigern werden Adressen an Unterprogramme übergeben.

**Anm.:**

`str[i]` ist äquivalent zu `*(str+i)`



## ÜBUNG: Call-by-reference, Umgang mit Matrizen

Schreiben Sie ein Unterprogramm, welches ein Feld von Integerwerten der Größe nach sortiert.

Ansatz: Durchlaufe das Feld immer wieder und vertausche jedesmal, wenn nötig, benachbarte Elemente.  
Der Vorgang kann abgebrochen werden, wenn bei einem Durchlauf kein Vertauschen mehr vorkommt (--> Bubblesort).

Weiter ist das Hauptprogramm zu schreiben, welches das Unterprogramm aufruft.



## ÜBUNG: Call-by-reference, Umgang mit Strings

Schreiben Sie ein Unterprogramm, welches zwei Strings lexikographisch vergleicht.

Ansatz: Die beiden Strings (s, t) werden Zeichen für Zeichen miteinander verglichen. Sobald ein Unterschied festgestellt wird, wird der Vorgang abgebrochen.

In diesem Fall wird die Differenz der ASCII-Werte der zuletzt betrachteten Zeichen ausgegeben.

Sind die beiden Strings gleich, wird 0 zurückgegeben.

Weiter ist das Hauptprogramm zu schreiben, welches das Unterprogramm aufruft.