

# Übungspaket 18

## Ausdrücke

---

### Übungsziele:

1. Ausdrücke in vielfältiger Form
2. Formulierung vereinfachender Kurzformen
3. Pre-/Post-Dekrement/Inkrement
4. Bedingte Auswertung
5. Besonderheiten logischer Ausdrücke

### Skript:

Kapitel: **43**

### Semester:

Wintersemester 2025/26

### Betreuer:

Benjamin, Thomas und Ralf

### Synopsis:

In Kapitel **43** haben wir gelernt, dass Ausdrücke in ganz unterschiedlichen Formen auftreten können. Die Programmiersprache C bietet verschiedene Kurzschreibweisen an, mittels derer sich Ausdrücke sehr kompakt formulieren lassen. Dies führt bei vielen Programmierern, insbesondere bei Anfängern, zumindest anfänglich zu Unverständnis, Frust und Ärger. Um diese Dinge zu vermeiden, schauen wir uns in diesem Übungspaket die verschiedenen Kurzformen und Besonderheiten systematisch an.

# Teil I: Stoffwiederholung

---

## Aufgabe 1: Kurzformen für Zuweisungen

Wie kann eine Zuweisung vereinfacht werden, wenn die selbe Variable sowohl auf der rechten als auch der linken Seite auftaucht?

Vereinfachung bei Verwendung des Operators op: `var = var op exp ⇒ var op= exp`

Notiere drei unterschiedliche Beispiele zur Illustration:

1. `i = i + 1 ⇒ i += 1`    2. `j = j * 3 ⇒ j *= 3`    3. `d=d/2.5 ⇒ d /= 2.5`

## Aufgabe 2: Die Pre- und Post-Operatoren

Zuerst sollten wir uns nochmals die hier relevanten Begriffe klar machen.

### 1. Pre-Inkrement:

Syntax	<code>++variable</code>
Beispiel	<code>++i</code>
Aktualisierungszeitpunkt	<i>vor</i> dem Variablenzugriff

### 2. Pre-Dekrement:

Syntax	<code>--variable</code>
Beispiel	<code>--i</code>
Aktualisierungszeitpunkt	<i>vor</i> dem Variablenzugriff

### 3. Post-Inkrement:

Syntax	<code>variable++</code>
Beispiel	<code>i++</code>
Aktualisierungszeitpunkt	<i>nach</i> dem Variablenzugriff

### 4. Post-Dekrement:

Syntax	<code>variable--</code>
Beispiel	<code>i--</code>
Aktualisierungszeitpunkt	<i>nach</i> dem Variablenzugriff

## Aufgabe 3: Bedingte Ausdrücke (Zuweisungen)

Zur Erinnerung: Hin und wieder will man einer Variablen einen Wert zuweisen, der wiederum von einer Bedingung abhängig ist. Beispiel: `if (condition) i = 1; else i = 2;` Je

nachdem, ob die Bedingung `condition` erfüllt ist oder nicht, wird der Variablen `i` der Wert 1 oder 2 zugewiesen. Etwas allgemeiner spricht man auch von bedingten Ausdrücken. Illustriere die Syntax der bedingten Zuweisung bzw. des bedingten Ausdrucks anhand zweier Beispiele:

Syntax	<code>condition? true_value: false_value</code>
Beispiel 1	<code>i = (j &gt; 3)? 1: -1</code>
Beispiel 2	<code>printf( "expression= %d\n", (j &gt; 3)? 1: -1 )</code>

## Aufgabe 4: Logische Ausdrücke

**Logische Werte:** Wie sind die beiden folgenden logischen Werte in der Programmiersprache C definiert?

wahr/true	alle Werte ungleich null
falsch/false	Wert gleich null

**Auswertung (Berechnung) logischer Ausdrücke:** Die Programmiersprache C wertet arithmetische und logische Ausdrücke grundsätzlich unterschiedlich aus. Bei arithmetischen Ausdrücken gilt, dass die Reihenfolge, in der die einzelnen Terme ausgewertet (berechnet) werden, dem Compiler überlassen ist. Der Compiler garantiert *nur*, dass er die Einzelteile gemäß der gültigen Rechenregeln korrekt zusammensetzt. Welche beiden Regeln gelten für die Auswertung logischer Ausdrücke?

Regel 1:	Logische Ausdrücke werden von links nach rechts ausgewertet.
Regel 2:	Das Auswerten wird abgebrochen, sobald das Ergebnis feststeht.

## Teil II: Quiz

---

### Aufgabe 1: Kurzformen für Zuweisungen

Gegeben sei folgendes Programm:

```
1 int main( int argc, char **argv )
2     {
3         int i = 1, j = 2, k = 3;
4         i += k; j -= k; k *= 3;
5         i *= j + k; j *= 100; j /= k; k -= j - i;
6         i -= (k = 4); j += 1;
7     }
```

Notiere in der folgenden Tabelle, welche Werte die einzelnen Variablen in den jeweiligen Programmzeilen annehmen:

Zeile	Werte von		
	i	j	k
3	1	2	3
4	4	-1	9
5	32	-11	52
6	28	-10	4

### Aufgabe 2: Bedingte Ausdrücke (Zuweisungen)

In den folgenden Anweisungen sind die beiden Variablen `i` und `j` vom Typ `int` und haben *immer* die Werte: `i = 1` und `j = 2`. Welche Werte haben sie anschließend?

Ausdruck	i	j
<code>i = (j == 2)? 1: 3;</code>	1	2
<code>i = (2)? 1: 3;</code>	1	2
<code>i = (0)? 1: 3;</code>	3	2
<code>i = (j = 1)? 5: 7;</code>	5	1
<code>i = (j = 1)? (j = 2) + 1: (j = 3) - 6;</code>	3	2
<code>i = (j++ &gt; 1)? 2: 4;</code>	2	3

### Aufgabe 3: Die Pre- und Post-Operatoren

Gegeben sei folgendes Programm:

```

1 #include <stdio.h>
2
3 int main( int argc, char **argv )
4     {
5         int i = 1, j = 2, k = 3;
6         printf( "i= %d j= %d k= %d\n", i++, ++j, k--);
7         printf( "i= %d j= %d k= %d\n", i++, ++j, k--);
8         printf( "i= %d j= %d k= %d\n", i++, ++j, k--);
9         printf( "i= %d j= %d k= %d\n", i, j, k);
10    }

```

Notiere in der folgenden Tabelle die Ausgaben für die einzelnen Variablen:

Ausgabe von			
Zeile	i	j	k
6	1	3	3
7	2	4	2

Ausgabe von			
Zeile	i	j	k
8	3	5	1
9	4	5	0

Im Folgenden werden die beiden Pre- und Post-Operatoren innerhalb etwas komplexerer Ausdrücke verwendet. Dabei sind die drei Variablen *i*, *j* und *k* vom Typ `int`, wobei am Anfang jeder Anweisung *immer* gilt: *i* = 1 und *j* = 2. Welche Ergebnisse liefern die folgenden Anweisungen?

Ausdruck	i	j	k	Bemerkung
<code>k = i++</code>	2	2	1	
<code>k = i--</code>	0	2	1	
<code>k = ++i</code>	2	2	2	
<code>k = --i</code>	0	2	0	
<code>k = i-- * ++j</code>	0	3	3	
<code>k = (1 + ++i)*j++</code>	2	3	6	
<code>k = (1 + ++i)*i</code>	2	2	?	<i>k</i> ist undefiniert, da die Evaluationsreihenfolge von <code>++i</code> und <code>i</code> undefiniert ist
<code>k = (1 + ++i)*i--</code>	1	2	?	<i>k</i> ist undefiniert, da die Evaluationsreihenfolge von <code>++i</code> und <code>i--</code> undefiniert ist

**Hinweis:** Bei der Beurteilung bzw. Auswertung obiger Ausdrücke sollte insbesondere Abschnitt [43.4](#) des Skriptes zu Rate gezogen werden.

## Aufgabe 4: Logische Ausdrücke

In den folgenden Aufgaben sind die beiden Variablen `i` und `j` vom Typ `int` und haben zu Beginn jeder Anweisung immer die Werte: `i = 1` und `j = 2`. Welche Ergebnisse liefern die folgenden Ausdrücke?

Ausdruck	Ergebnis	i	j
<code>i == 1</code>	1	1	2
<code>i &gt; -2</code>	1	1	2
<code>i == 1 &amp;&amp; j != 0</code>	1	1	2
<code>i &amp;&amp; j</code>	1	1	2
<code>i++ &amp;&amp; j++</code>	1	2	3
<code>i--    j++</code>	1	0	2
<code>i++ &gt; 2 &amp;&amp; j++</code>	0	2	2

## Aufgabe 5: Verschachtelte Zuweisungen

In den folgenden Beispielen sind alle Variablen vom Typ `int`. Welche Werte haben die Variablen nach den Zuweisungen?

Ausdruck	i	j
<code>i = (j = 5)</code>	5	5
<code>i = (j = 5) &lt; 10</code>	1	5
<code>i = j = 5 &lt; 10</code>	1	1
<code>i = (j = 5)*(i = 2)</code>	10	5
<code>i = ((j = 5) &gt; 3) * 2</code>	2	5

# Teil III: Fehlersuche

---

## Aufgabe 1: Fehler in Kurzformen

Beim folgenden (sinnlosen) Programm hat DR. CHATTY wieder Einiges falsch gemacht. Finde und korrigiere die vorhandenen Syntaxfehler. Manchmal gibt es mehrere Korrekturmöglichkeiten. Entscheide dich für eine; Hauptsache jede Zeile wird korrekt.

```
1 int main( int argc, char **argv )
2     {
3         int i = 1--, j = -2;
4         i *= += 1;
5         i *= += j;
6         i **= 3;
7         j = ++i++;
8         j = 3 *= i;
9         j -= 3 * (i += 2),
10        i++ = 3 * j;
11        i-- += 3;
12    }
```

Zeile	Fehler	Erläuterung	Korrektur
3	1--	Die Pre- und Post-Increment/Decrement Operatoren kann man nur auf Variablen und nicht auf Konstanten anwenden; es wäre sinnlos, die 1 auf null zu reduzieren.	i = 1
4	*= bzw. +=	Bei einer Zuweisung kann man nur <i>einen</i> dieser Kurzoperatoren und nicht mehrere gleichzeitig anwenden. Wir entscheiden uns für den zweiten.	i += 1
5	dito.	Da rechts aber eine Variable steht, könnte man die Ausdrücke schachteln, denn der Wert einer Zuweisung ist das Ergebnis. Das ++ bezieht sich auf j, i wird um den neuen Wert von j erhöht. Einfach mal probieren.	i *= (++j)
6	**=	Bei der Kurzfassung kommt erst das „Rechensymbol“ und dann das Gleichheitszeichen.	*=
7	++i++	Bei <i>einer</i> Variablen ist <i>entweder</i> eine Pre- oder eine Post-Operation anwendbar. Wir nehmen letzteres.	i++
8	*=	Vermutlich ist einfach nur das = zu viel.	j = 3 * i
9	, statt ;	Eigentlich ist alles richtig! Hinten fehlt aber ein ;	;

Zeile	Fehler	Erläuterung	Korrektur
10	++ und =	Es geht nicht beides gleichzeitig.	<code>i = 3*j+1</code>
11	-- +=	dito.	<code>i += 3 - 1</code>

### Programm mit Korrekturen:

```

1 int main( int argc, char **argv )
2     {
3         int i = 1, j = -2;
4         i += 1;
5         i *= ++ j;
6         i *= 3;
7         j = i++;           // oder j = ++i;
8         j = 3 * i;
9         j -= 3 * (i += 2);
10        i = 3 * j + 1;
11        i += 3 - 1;
12    }

```

# Teil IV: Anwendungen

---

## Aufgabe 1: Kurzformen

Vereinfache die folgenden Ausdrücke soweit möglich. Beachte, dass es manchmal unterschiedliche Lösungen gibt. Bei Unsicherheiten oder Zweifeln die Ergebnisse einfach mittels eines Mini-C-Programms überprüfen!

Ausdruck	Kurzform	Alternativen
<code>k = k + 1</code>	<code>k += 1</code>	<code>k -= -1</code> , <code>k++</code> , <code>++k</code>
<code>k = k * 2</code>	<code>k *= 2</code>	<code>k += k</code>
<code>k = k - 4711</code>	<code>k -= 4711</code>	<code>k += -4711</code>
<code>k = k - 23 - i</code>	<code>k -= 23 + i</code>	<code>k += -23 - i</code>
<code>k = 2 * k - i - 1</code>	<code>k *= 2</code> ; <code>k -= i + 1</code>	<code>k += k - i - 1</code> oder <code>k -= -k + i + 1</code>
<code>d = (123.0 + 0.123) * d</code>	<code>d *= 123.0 + 0.123</code>	<code>d *= 123.123</code>
<code>i = i / 123</code>	<code>i /= 123</code>	<code>i *= 1/123</code> funktioniert nicht, da dies identisch mit: <code>i *= 0</code> ist

## Aufgabe 2: Vereinfachung logischer Ausdrücke

Mit etwas Übung lassen sich logische Ausdrücke ein klein wenig vereinfachen. Dies üben wir anhand von vier unterschiedlichen Aufgaben. Fülle zunächst die Wahrheitstabelle aus und überlege dir dann eine mögliche Vereinfachung. In den folgenden Tabellen steht `a` immer für eine Bedingung, die die Werte `wahr` (1) und `falsch` (0) annehmen kann.

a „und“ wahr:	a „und“ falsch:	a „oder“ wahr:	a „oder“ falsch:
<u>a a &amp;&amp; 1</u>	<u>a a &amp;&amp; 0</u>	<u>a a    1</u>	<u>a a    0</u>
0 0	0 0	0 1	0 0
1 1	1 0	1 1	1 1
Vereinfachung <input type="checkbox"/> a	Vereinfachung <input type="checkbox"/> 0	Vereinfachung <input type="checkbox"/> 1	Vereinfachung <input type="checkbox"/> a

### Zusammenfassung:

logisches und:	<input type="text" value="a „und“ wahr ergibt a, a „und“ falsch ergibt falsch."/>
logisches oder:	<input type="text" value="a „oder“ wahr ergibt wahr, a „oder“ falsch ergibt a."/>