Herzlich willkommen!

Dozent: Dipl.-Ing. Jürgen Wemheuer

Teil 3: Ausdrücke und Anweisungen

Mail: wemheuer@ewla.de
Online: http://cpp.ewla.de/

Ausdrücke und Anweisungen (1)

1. Anweisung: Sie
   - steuert die Reihenfolge der Ausführung eines Ausdrucks,
   - wertet ihn aus oder
   - bewirkt nichts.

- Eine einfache Anweisung ist die Zuweisung: x = a + b ;
  Anweisungen enden immer mit einem Semikolon!

  Beispielinterpretation: nicht „x ist gleich a plus b“, sondern eher:

  „Weise der Summe aus a und b den Wert x zu“,

  „x erhält den Wert aus der Summe von a und b“

  „Hole Wert a, hole Wert b, addiere beide Werte und speichere das Ergebnis auf dem Speicherplatz, der für die Variable x zugewiesen ist“

- Whitespace (Leerzeichen):
  - Leerzeichen, Tabulatoren und Zeilenvorschübe ignoriert der Compiler:
  - x = a + b; <=> x = a + b; <=> x= a+ b;
  - Hinweis: Mit Whitespace den Quellcode übersichtlich gestalten!

Ausdrücke und Anweisungen (2)

2. Verbundanweisungen:

Verbundanweisungen sind zusammengesetzte Anweisungen, die in geschweifte Klammern eingeschlossen werden:

```
{Anweisung; Anweisung; Anweisung;...;}
```

Nach der Verbundanweisung entfällt das Semikolon!

Hinweis: Nach der öffnenden geschweiften Klammer "{" immer gleich die schließende geschweifte Klammer "}" schreiben und dann "mitschieben" lassen.

Ausdrücke und Anweisungen (2)

2. Ausdruck: liefert einen Wert zurück, der weiterverarbeitet werden kann.
Wenn also etwas einen Wert hat, dann ist es ein Ausdruck.
Alle Ausdrücke sind Anweisungen.

Beispiel:
Der Ausdruck **u = pi * d** multipliziert **pi** mit **d** und liefert den Wert dieser Berechnung an **u**.

Diese Anweisung ist ebenfalls wieder ein Ausdruck, denn **u** liefert einen Wert - dieser Wert kann wieder weiterverarbeitet, z.B. zugewiesen werden:

```
L = u = pi * d
```

Beispiel:

```
float L=0; u=0; d=9; pi=3.14159;
cout << „pi: „ << pi << „, d: „ << d << endl;
L = u = pi * d;
cout << „L: „ << L << „, u: „ << u << endl;
```

Operatoren

Ein Operator ist ein Symbol (=, +, -, <<, *, ==, <, …), das den Computer zur Ausführung einer damit fest vereinbarten Aktion veranlasst.

1. Der Zuweisungsoperator =

Beispiel: u = pi * d;

Der Operand auf der linken Seite (L-Wert) u bekommt durch den Zuweisungsoperator = den Wert von der rechten Seite (R-Wert) zugewiesen.

(L-Wert: Operand, der auf der linken Seite einer Zuweisung = steht)

(R-Wert: Operand, der auf der rechten Seite einer Zuweisung = steht)

Alle L-Werte sind immer auch als R-Werte zulässig,
aber nicht immer auch umgekehrt!

Ein Literal ist ein R-Wert, der nicht als L-Wert gesetzt werden darf / kann:

```
pi = 3.14159; // zulässig
3.14159 = pi; // nicht zulässig
```

Operatoren

2. Mathematische Operatoren:

Addition **+** Grundrechenart für alle Variablentypen definiert

Subtraktion **-** Grundrechenart für alle Variablentypen definiert

Multiplikation ***** Grundrechenart für alle Variablentypen definiert

Division **/** Grundrechenart, für Integer nur mit ganzer Zahl ohne Rest definiert: 50/7 -> 7, der Rest (1) wird abgeschnitten

Modulo-(Restwert-)Division **%** gibt den Rest einer Ganzzahldivision zurück: 50%7 -> 1 (50/7 = 7 Rest 1)

Operatoren

3. Zusammengesetzte Operatoren:

Das Ergebnis einer mathematischen Operation mit einer Variablen soll wieder genau dieser Variablen zugewiesen werden:

```
z = u + 9;
u = z;
```

kürzer:

```
u = u + 9; //Zuweisung, kein „gleich“
```

In C++ noch kürzer mit zusammengesetztem Operator:

```
u += 9; // lies: u plus gleich 9
```

Operator	Symbolik	bedeutet
Plus-gleich	+=	u = u + x
Minus-gleich	-=	u = u - x
Mal-gleich	*=	u = u * x
Geteilt-gleich	/=	u = u / x
Modulo-gleich	%=	u = u % x

Inkrementierung / Dekrementierung

Erhöhen des Werts einer Variablen um 1 -> Inkrementieren
Erniedrigen des Werts einer Variablen um 1 -> Dekrementieren

Der Inkrement-Operator **++** erhöht den Wert einer Variablen um 1

Der Dekrement-Operator **--** erniedrigt den Wert einer Variablen um 1

```
i = i + 1 -> i += 1 -> i++
```

Präfix-Variante: **++i** oder **--i** // Auswertung VOR der Zuweisung

Postfix-Variante: **i++** oder **i--** // Auswertung NACH der Zuweisung

```
int x=8;
y = ++x; // 1. x=x+1 => 9  2. y=x => 9
z = x++; // 1. z=x => 9    2. x=x+1 => 10
```

Ausdrücke und Anweisungen (6)

6. Verschachtelte Klammern:

Verschachtelungen (Klammer-in-Klammer-Ausdrücke) sind sogenannte komplexe Ausdrücke und immer von innen nach außen zu lesen und abzuarbeiten

Verschachtelungen lassen sich mit etwas mehr Schreibarbeit vermeiden, Beispiel:

Berechnung Personen_km_proBus für 2 Busunternehmen mit 15 Bussen

```
Personen_km_proBus = ((Personen01+Personen02) * (km01+km02)/15;
```

```
Personen = Personen01 + Personen02;
km = km01 + km02;
Personen_km = Personen * km;
Personen_km_proBus = Personen_km / 15;
```

Ausdrücke und Anweisungen (7)

7. Wahrheitswerte:

Jeder Ausdruck kann auf seinen Wahrheitswert geprüft werden:

Hat ein Ausdruck den mathematischen Wert Null, dann ist sein Wahrheitswert falsch (false), sonst immer wahr (true)

Der Datentyp bool (zwei Werte true oder false) entspricht dem Wahrheitswert nach ISO/ANSI

(ISO = International Standards Organisation)
(ANSI = American National Standards Institute)

8. Vergleichsoperatoren:

Vergleichsoperatoren sind der bestimmende Bestandteil eines Vergleichsausdrucks

Der Vergleichsausdruck vergleicht immer Zahlenwerte und liefert immer nur einen der beiden Werte **true** oder **false**

Bei anderen Datentypen (z.B. char) wird der Typ konvertiert!

Es gibt folgende Vergleichsoperatoren:	
Gleich	==
Ungleich	!=
Größer als	>
Größer oder gleich	>=
Kleiner als	<
Kleiner oder gleich	<=

Ausdrücke und Anweisungen (9)

9. Logische Operatoren:

Werden benötigt, wenn für eine Vergleichsoperation mehrere Vergleiche (logisch) miteinander verknüpft werden sollen.

Die Verknüpfung mehrerer Vergleiche liefert ebenfalls immer nur einen der beiden Werte **true** oder **false**

Es gibt folgende logische Operatoren in C++:

Operator	Symbol	Bedeutung	Beispiel
AND	`&&`	Logisch UND	`Ferien && schoenesWetter`
OR	`||`	Logisch ODER	`Sekt || Selters`
NOT	`!`	Logisch NICHT	`!Unterrichtsausfall`

10. Rangfolge:

Für die Abarbeitungs-Reihenfolge aller Operatoren in C++ gibt es festgesetzte Vorrangregeln – diese sind in den C++-Dokumentationen nachzulesen. Man muss diesen Sachverhalt kennen – aber nicht auswendig.

Operatoren mit höherem Vorrang (kleinere Zahl!) werden vor Operatoren mit niedrigerem Vorrang (höhere Zahl!) abgearbeitet.

Haben Operatoren den gleichen Rang, werden sie in der Reihenfolge von links nach rechts ausgeführt.

Soll die Reihenfolge nicht nach den Vorrangregeln ausgeführt werden, setzt man runde Klammern und erreicht damit eine Änderung der Rangfolge.

Beispiel: Pfad_n in [m] (Meter) und Wanderweg in [km] (Kilometer):

```
Wanderweg =  Pfad_1 + Pfad_2 + Pfad_3  / 1000;
Wanderweg = (Pfad_1 + Pfad_2 + Pfad_3) / 1000;
```

Programmverzweigung: if (1)

Abhängig vom Wahrheitsgehalt eines Ausdrucks findet eine Verzweigung innerhalb des Quellcodes statt.

Der Wahrheitsgehalt (das Ergebnis einer Vergleichsoperation) kann immer nur wahr oder falsch sein:

Beispiel 1 : einfachste if-Anweisung

```
if (Ausdruck)
  Anweisung;
```

```
if ( x==9 )
  Wert = 100;
```

- ist die Bedingung erfüllt (boolscher Wert: true), wird die Anweisung ausgeführt
- ist die Bedingung nicht erfüllt (boolscher Wert: false), wird die Anweisung übersprungen

Mit der else-Klausel kann zusätzlich ein Quellcode angegeben werden, der bei Nichterfüllung der Bedingung auszuführen ist:

Beispiel 2 : einfache if-Anweisung mit else-Klausel

```
if (Ausdruck)
  Anweisung1;
else
  Anweisung2;
```

```
if ( x==9 )
  Wert = 100;
else
  Wert = Wert +9;
```

Programmverzweigung: if (2)

Umfasst ein Programmzweig mehrere Befehle,
sind unbedingt geschweifte Klammern zu verwenden!

```
if (x==9)                                   // OK...
  Wert = 100;                               // die “true-Anweisung“
  cout << “Wert ist jetzt 100“;             // wird IMMER ausgeführt
else                                        // Fehler: else ohne if
  Wert = Wert + 9;
  cout << “Wert wurde um 9 erhoeht“;
cout << Wert;


if (x==9)     {                             // OK...
  Wert = 100;                               // die “true-Anweisung“
  cout << “Wert ist jetzt 100“;}            // geht bis hierher
else          {                             // OK
  Wert = Wert + 9;
  cout << “Wert wurde um 9 erhoeht“;}
cout << Wert;
```

Programmverzweigung: if (3)

Mit geschweiften Klammern lassen sich sehr komplexe if-Anweisung erzeugen – die Verschachtelung kennt keine Grenzen…

```
if (Ausdruck1)
{
        if (Ausdruck2)
        {
                Anweisung1;
                Anweisung2;
        }
        else
        {
                if (Ausdruck3)
                        Anweisung3;
                else
                        Anweisung4;
        }
}
else
        Anweisung5;
```

Programmverzweigung: switch (1)

Die **if**-Anweisung wertet immer nur **einen** Ausdruck aus,
die **switch**-Anweisung lässt eine **Mehrfach-Verzweigung** zu,
abhängig von mehreren unterschiedlichen Werten:

```
switch (Ausdruck) {
   case Wert1: Anweisung1;
      break;
   case Wert2: Anweisung2a;
      Anweisung2b;
      break;
…
   default: AnweisungX;
}
```

Ausdruck: Jeder (noch so komplexe) C++-Ausdruck ist gültig
Anweisung: Jede gültige C++-Anweisung / Anweisungsblock

Programmverzweigung: switch (2)

Das Programm „schaut“ auf die erste Übereinstimmung von Ausdruck und Wert und setzt hier den Ablauf fort.

Deshalb immer darauf achten, dass ein **`break`** vorhanden ist!

Die **`default`**-Anweisung ist optional, sollte aber immer gesetzt werden – und wenn auch nur zur Fehlermeldung…

Programmverzweigung: switch (3)

Ein Beispiel für die Anwendung der switch-Anweisung:

```
int main() {
  unsigned short int Note;
  cout << „Bitte eine Note von 1 bis 6 eingeben: „;
  cin >> Note;
  switch (Note) {
    case 0: cout << “Leider zu klein!“; break;
    case 1: cout << “Sehr gut!“; break;
    case 2: cout << “Gut“; break;
    case 3: cout << “Befriedigend“; break;
    case 4: cout << “Ausreichend“; break;
    case 5:
    case 6: cout << “Nicht bestanden!“; break;
    default: cout << “Eingabe ungueltig“;
  }
}
```

Verkürzte if-Notation

```
Wert = ( (x==9) ? 100 : (Wert+9) );
```

```
if (x==9) Wert=100; else Wert=Wert+9;
```