Programmieren 1

Wintersemester 2016/2017

Marcus Riemer, B.Sc.

Basierend auf den Unterlagen „Programmstrukturen 1“
von Prof. Dr. Andreas Häuslein

3 Konzept der Konstanten

- Zusätzliche Anforderung: Berücksichtigung der Mehrwertsteuer
- Erste Modifikation des Programms:

```
program Viertes;
var
   Menge : Word;
   Preis : Real;
   Nettobetrag,
   Bruttobetrag : Real;

begin
   write ('Bitte geben Sie die Menge ein: ');
   readln (Menge);
   write ('Bitte geben Sie den Preis ein: ');
   readln (Preis);
   Nettobetrag := Menge * Preis;
   writeln ('Der Nettobetrag ist: ',Nettobetrag);
   Bruttobetrag := Nettobetrag + Nettobetrag * 16/100;
   writeln ('Der Bruttobetrag ist: ', Bruttobetrag);
   writeln ('MwSt-Satz: 16%')
end.
```

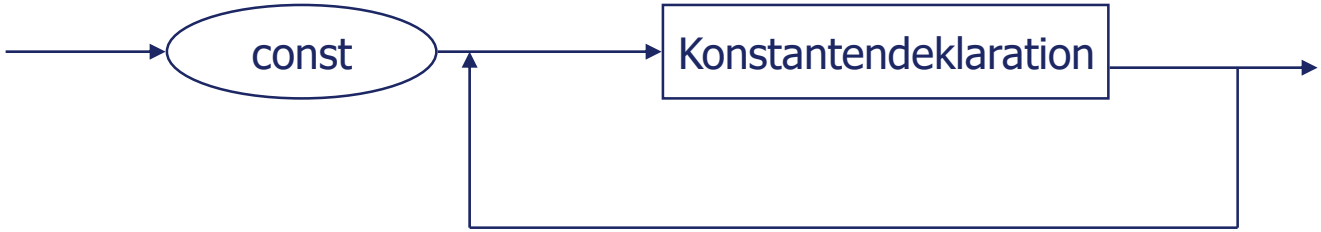
Direkte Angabe von Werten in Programmen:
Problem der Änderbarkeit!
(hier: Aktualisierung Mehrwertsteuersatz)

2 Konzept der Konstanten

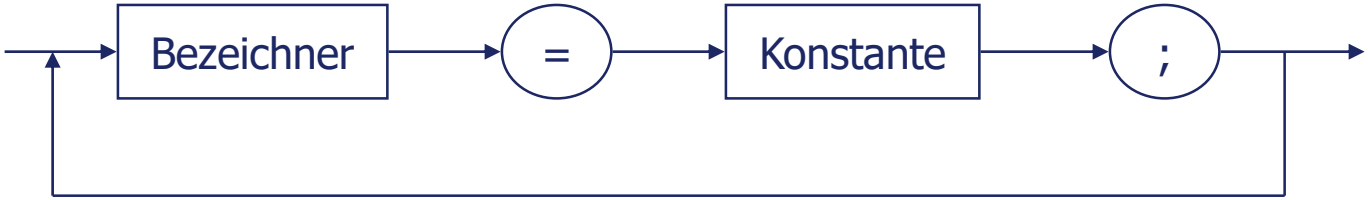
- Konstanten sind benannte Datenobjekte: Einem Wert wird ein Bezeichner zugeordnet
- Verwendung des Bezeichners überall dort, wo Wert im Programm benötigt wird
- Gleiche Wirkung wie direkte Angabe des Wertes
- Verbesserung der Änderbarkeit
  - Reduzierung des Änderungsaufwandes
  - Gewährleistung der Konsistenz
- Verbesserung der Lesbarkeit
- Wert von Konstanten wird *zur Übersetzungszeit* des Programms ermittelt und festgelegt
- Konstanten sind zur Laufzeit des Programms nicht veränderbar
- Konstanten müssen vor ihrer Verwendung deklariert werden

2 Konzept der Konstanten

Konstantendeklarationen



Konstantendeklaration



Konstante

Wert des Ausdrucks muss zur Übersetzungszeit zu ermitteln sein

3 Konzept der Konstanten

- Beispiele für Konstantendeklarationen:

```
const
      Monate = 12;
      Anzahl_Arbeitstage = 5;

      Urlaubstage = 32;
      Alterszuschlag = 2;
      Urlaubsanspruch = Urlaubstage + Alterszuschlag;

      Normaltemperatur = 36.5;
      EndeZeichen = 'E';
      Fertig = True;
      Meldung = 'Eingabefehler!';
```

3 Konzept der Konstanten

- Anwendung auf Programmbeispiel:

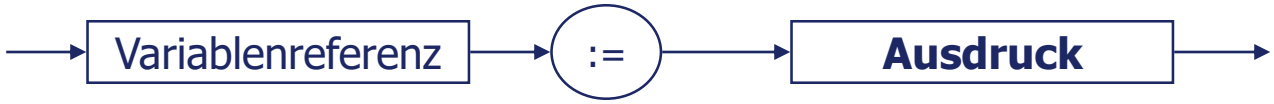
```
program Fuenftes;

const MWST_SATZ = 16;

var
  Menge : Word;
  Preis : Real;
  Nettobetrag,
  Bruttobetrag : Real;

begin
  write ('Bitte geben Sie die Menge ein: ');
  readln (Menge);
  write ('Bitte geben Sie den Preis ein: ');
  readln (Preis);
  Nettobetrag := Menge * Preis;
  writeln ('Der Nettobetrag ist: ', Nettobetrag);
  Bruttobetrag := Nettobetrag + Nettobetrag * MWST_SATZ /100;
  writeln ('Der Bruttobetrag ist: ', Bruttobetrag);
  writeln ('MwSt-Satz: ', MWST_SATZ, '%');
end.
```

Mehrere (beliebig viele) Auswirkungen!

1 Änderung

3 Ausdrücke

- Ausdrücke gehören zu den Grundelementen von Programmen (neben Anweisungen)
- Ausdrücke werden durch Kombination von beliebig vielen Operatoren und Operanden gebildet
- Ein Ausdruck beschreibt, welche Operatoren auf welche Operanden angewendet werden sollen
- Die Anwendung der Operatoren wird *Auswertung eines Ausdrucks* genannt
- Als Ergebnis der Auswertung eines (korrekten) Ausdrucks ergibt sich ein Ergebniswert
- Anstelle einer direkten Wertangabe oder einer einzelnen Variablen sind im Programm meist auch komplexere Ausdrücke zulässig
- Beispiel: die rechte Seite einer Zuweisung zu einer Variablen:



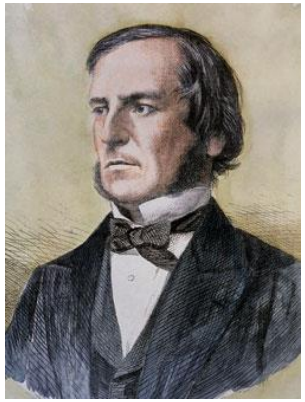
3 Binäre arithmetische Operatoren

Operator	Operation	Operandentyp	Ergebnistyp
+	Addition	Ganzzahlig	Ganzzahlig
+	Addition	Mind. 1 Operand Reellwertig	Reellwertig
-	Subtraktion	Ganzzahlig	Ganzzahlig
-	Subtraktion	Mind. 1 Operand Reellwertig	Reellwertig
*	Multiplikation	Ganzzahlig	Ganzzahlig
*	Multiplikation	Mind. 1 Operand Reellwertig	Reellwertig
/	Division	Ganzzahlig	Reellwertig
/	Division	Reellwertig	Reellwertig
div	ganzzahlige Division	Ganzzahlig	Ganzzahlig
mod	ganzzahliger Divisionsrest	Ganzzahlig	Ganzzahlig

- `2 + 2`
- `2.0 * 2`
- `2 - 2.0`
- `2 / 2`
- `2 div 2`
- `2.0 div + 2.0`

3 Logische Operatoren

- Verfügbare Operatoren: `not`, `and`, `or`, `xor`
- Auch als *boolesche* Operatoren bezeichnet
- Zulässige Operandentypen:
  - Beide Operanden *boolesche Werte* (Normalfall): Übliche logische Verknüpfung der Operanden
  - Beide Operanden *Ganzzahlwerte* (Ausnahmefall): Verknüpfung der einzelnen Bits der Operanden



George Bool
(1815 – 1864)

p	q	p **and** q	p **or** q	p **xor** q
true	true	true	true	false
true	false	false	true	true
false	true	false	true	true
false	false	false	false	false

p	**not** p
true	false
false	true

3 Relationale Operatoren

Operator	Operation	Operandentyp	Ergebnistyp
=	gleich	einfacher Datentyp	Boolean
<>	ungleich	einfacher Datentyp	Boolean
<	kleiner	einfacher Datentyp	Boolean
>	größer	einfacher Datentyp	Boolean
>=	größer gleich	einfacher Datentyp	Boolean
<=	kleiner gleich	einfacher Datentyp	Boolean

- Operanden der Vergleichsoperatoren müssen kompatibel sein
  - Zahlen können nur mit Zahlen verglichen werden (auch ganze Zahlen mit Gleitkommazahlen)
  - Zeichen können nur mit Zeichen verglichen werden

3 Auswertungsreihenfolge

- Auswertungsreihenfolge: Reihenfolge, in der Operatoren auf Operanden bzw. Zwischenergebnisse angewendet werden
- Auswertungsreihenfolge wird bestimmt durch
  - Reihenfolge im Text des Ausdrucks (von links nach rechts)
  - Klammerung
  - Rangfolge der Operatoren
- Rangfolge der Operatoren ist durch Prioritäten für Operatorkategorien festgelegt:

Operatoren	**Priorität**	**Kategorie**
not, +, -	1. (hoch)	unäre Operatoren
* , / , div, mod, and	2.	multiplikative Operatoren
+ , - , or , xor	3.	additive Operatoren
= , < , <= , > , >= , <>	4. (niedrig)	relationale Operatoren

3 Zusammenfassung

Ergebnistyp

- Für `Addition`, `Subtraktion` und `Multiplikation` gilt: Ist einer der Operanden eine reelle Zahl, so ist auch das Ergebnis eine reelle Zahl.
- Für `Division` gilt: Der Ergebnistyp ist unabhängig von den Typen der Operanden stets reell.
- Für die `ganzzahlige Division` gilt: Beide Operanden müssen ganzzahlig sein, das Ergebnis ist ebenfalls immer ganzzahlig.

Auswertungsreihenfolge

Operatoren	**Priorität**	**Kategorie**
not, +, -	1. (hoch)	unäre Operatoren
* , / , div, mod, and	2.	multiplikative Operatoren
+ , - , or , xor	3.	additive Operatoren
= , < , <= , > , >= , <>	4. (niedrig)	relationale Operatoren

3 Übung "Auswertungsreihenfolge und Ergebnistypen"

- Klammern und vereinfachen Sie die folgenden Ausdrücke gemäß den Regeln der Auswertungsreihenfolge.
- Werten Sie den Ausdruck aus und geben Sie außerdem den Typ des Ergebnisses an.
- Einige dieser Ausdrücke sind nicht gültig, geben Sie in diesem Fall die Typ-Inkompatabilität an.

Beispiele

a) **`true and false or true`**

```
→ (true and false) or true
→ false or true
→ true (boolean)
```

b) **`false or false and true`**

```
→ false or (false and true)
→ false or false
→ false (boolean)
```

c) **`2 >= 3 - 2`**

```
→ 2 >= (3 - 2)
→ 2 >= 1
→ true (boolean)
```

d) **`not 2.0 >= 3`**

```
→ (not 2.0) >= 3
→ Fehler, 2 ist eine Fließkommazahl
```

3 Übung “Auswertungsreihenfolge und Ergebnistypen”

```
a)  true and true and false or true
b)  not false and not false and not true or not false
c)  12 / 2
d)  12 div 2
e)  2 mod 12
f)  2 mod 12.0
g)  3 + 2 / 4 mod 2
h)  5 + 3 * 6 mod 2
i)  4 / 4 >= 1 * 2 + 2
j)  'a' >= 2 + 2
k)  'a' = 'a' and 2 >= 2
l)  ('a' = 'a') and (2 >= 2)
m)  4 + 2 mod 7 * 3
```

Wir programmieren ab sofort in der Delphi XE2 Entwicklungsumgebung

- Windows-Programm zur Entwicklung von Programmen
- Anzeige von Fehlern, Code-Vervollständigung, Debugging, ...

Praktische Einführung am Rechner

3 Übung “Auswertungen in einem Pascal-Programm”

```
// Demonstriert die Auswertung von Ausdrücken in Pascal
// Marcus Riemer, 12.11.2016
program auswertung;

begin
   writeln('a) ', true and true and false or true);
   writeln('b) ', not false and not false and not true or not false);
   writeln('c) ', 12 / 2);
   writeln('d) ', 12 div 2);
   writeln('e) ', 2 mod 12);
// writeln('f) ', 2 mod 12.0);
// writeln('g) ', 3 + 2 / 4 mod 2);
   writeln('h) ', 5 + 3 * 6 mod 2);
   writeln('i) ', 4 / 4 >= 1 * 2 + 2);
// writeln('j) ', 'a' >= 2 + 2);
// writeln('k) ', 'a' = 'a' and 2 >= 2);
   writeln('l) ', ('a' = 'a') and (2 >= 2));
   writeln('m) ', 4 + 2 mod 7 * 3);
end.
```

3 Übung “Treueprogramm” – Wiederholung

Aufbauend auf der Übung “Einfache Rechnung” wollen wir diese nun um ein einfaches Treueprogramm erweitern.

```
// Einfaches Rechnungsprogramm für zwei Produkte
// Birger Wolter, 8.11.2016
program umsatz;

var
  preis1, preis2, anzahl1, anzahl2, gesamtpreis : integer;
begin
  preis1 := 15;
  preis2 := 412;
  write('Anzahl Flanschmuffen: ');
  readln(anzahl1);
  write('Anzahl Adapter: ');
  readln(anzahl2);
  gesamtpreis := preis1 * anzahl1 + preis2 * anzahl2;
  writeln('Gesamtpreis ist ', gesamtpreis, ' Cent')
end.
```

3 Übung “Treueprogramm” – Konstanten und Format

Achtung: Für die Lösung dieser Aufgabe dürfen Sie keine bedingten Anweisungen (also kein `if`) verwenden.

a) Führen Sie Konstanten für die Preise der beiden Artikel ein. Nutzen Sie diese für eine Ausgabe der aktuellen Preise bevor der Benutzer eine Mengen-Eingabe tätigt und für die spätere Berechnung.

b) Geben Sie den errechneten Preis im folgenden Format aus, nutzen sie dafür die Operatoren `div` und `mod`. Speichern Sie die berechneten Werte für Euro und Cent vor der Ausgabe in einer entsprechenden Variable, diese werden für die weitere Ausführung des Programms benötigt.

```
120  => 1 Euro und 20 Cent
0    => 0 Euro und 0 Cent
1250 => 12 Euro und 50 Cent
```

3 Übung “Treueprogramm” – Stempel und Marken

c) Bei jedem Kauf können Treuemarken erworben werden. Diese unterscheiden sich in Stempel (Wahrheitswert, bei `true` wird gestempelt) und Sammelmarken (Anzahl der herausgegebenen Marken hängt vom Einkauf ab). Für Sammelmarken soll die für eine Marke benötigte Anzahl an Produkten ausgegeben werden.

1) Stempel “Einkauf”: Der Kunde hat überhaupt etwas gekauft.
2) Stempel “Krösus”: Der Kunde tätigt einen Einkauf für mehr als 1000€.
3) Stempel “Merkwürdig spezifisch”: Der Kunde kauft exakt 12 Flanschmuffen und 42 Adapter.
4) Stempel “Das war knapp”: Der Kunde muss unabhängig von der Anzahl an ganzen Euros 99 Cent bezahlen.
5) Sammelmarke “Etel-Tuning”: Eine Marke je 10 gekaufte Adapter “Drehstrom <-> Gardena”.
6) Sammelmarke “Reinhard Mey”: Eine Marke je 12 gekaufter Flanschmuffen.

3 Übung “Treueprogramm” – Beispielhafte Ausgabe

```
Eine Flanschmuffe kostet 140 Cent
Ein Adapter Drehstrom-Gardena kostet 220 Cent.
Anzahl Flanschmuffen: 12
Anzahl Adapter: 42
Der Preis beträgt 109 Euro und 20 Cent.
Stempel “Einkauf”: TRUE
Stempel “Krösus”: FALSE
Stempel “Merkwürdig Spezifisch”: TRUE
Stempel “Das war knapp”: FALSE
1 Marken “Etel-Tuning” (1 Marke je 10 Adapter)
3 Marken “Reinhard Mey” (1 Marke je 12 Flanschmuffen)
```

3 Übung “Treueprogramm” – Besprechung

- Vier Konstanten:
```
PREIS_FLANSCH = 15; PREIS_ADAPTER = 412;
SAMMEL_JE_FLANSCH = 12; SAMMEL_JE_ADAPTER = 10;
```
- Fünf Variablen:
```
anzFlansch, anzAdapter : Integer;
preisCentGesamt : Integer;
preisEuro, preisCent : Integer;
```
- Preisberechnung:
```
preisCentGesamt := anzFlansch * PREIS_FLANSCH +
                   anzAdapter * PREIS_ADAPTER;
preisEuro := preisCentGesamt div 100;
preisCent := preisCentGesamt mod 100;
```
- Berechnungen:
```
writeln('Stempel "Das war knapp": ', preisCent = 99);
writeln(anzFlansch div SAMMEL_JE_FLANSCH,
        ' Sammelmarke "Etel-Tuning"');
```