Boolean

- Wertemenge: Wahrheitswerte {FALSE,TRUE}, auch {0,1}
- Deklaration:

```
VAR  present,billig,laut,gefunden : BOOLEAN;
```

- Ein-/Ausgabe:

```
keine!
```

- Operatoren:
  - Negation, Verneinung NOT ~
  - Konjunktion, logisches UND AND &
  - Disjunktion, logisches ODER OR

- Vergleichsoperationen
  - Gleichheit =
  - Ungleichheit <> #
  - Mengenzugehörigkeit? IN
- Operatoren sind zweistellig
  - Beispiel: (x<y) AND (y<z)
  - falsch: x<y<z

Boolean (cont'd)

- Verwendung
  - Ausdruck einer erfüllten oder nicht erfüllten Eigenschaft
  - Variablen vom Typ BOOLEAN sollten Eigenschaftsworte sein
- Boolean in Bedingungen:

```
          IF billig THEN ... ELSE ...
anstatt:  IF billig = TRUE THEN ... ELSE ...

          IF NOT billig THEN ... ELSE ...
anstatt:  IF billig = FALSE THEN ... ELSE ...
```

Wahrheitstabelle:

P	NOT p
0	1
1	0

Wahrheitstabelle:

p	q	p AND q	p OR q
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

de Morgan‘sche Regeln:

NOT (p AND q) = (NOT p) OR (NOT q)

NOT (p OR q) = (NOT p) AND (NOT q)

Vergleiche:

p	q	p < q	p <= q	p > q	p >= q
1	1	0	1	0	1
1	0	0	0	1	1
0	1	1	1	0	0
0	0	0	1	0	1

Implikation:

p	q	p → q	p <= q	NOTp OR q
1	1	1	1	1
1	0	0	0	0
0	1	1	1	1
0	0	1	1	1

Character

- standardisierter Zeichensatz
  - Festlegung einer rechnerinternen Darstellung von Zeichen (Bitmuster)
  - Festlegung einer Ordnung über der Zeichenmenge
  - Beispiel: ASCII (American Standard Code for Information Interchange)
    - M ist größer als !, denn (M ↔ 33D ↔ 21H), (! ↔ 77D ↔ 4DH)
- Deklaration:

```
VAR  ch1,ch2,ch3 : CHAR;
```

- Zuweisung:

```
ch1 := "a";
ch2 := 'b';
ch3 := 65C;
```

(65C ist ASCII-Oktal-Code für 5 (↔ 53D ↔ 35H))

- Ein-/Ausgabe:

```
FROM InOut IMPORT Read,Write;
```

- Operationen: Vergleiche auf Basis der ASCII-Ordnung
  - ORD(ch) liefert den ASCII-Wert (Ordnungsnr.) des Characters ch
  - CHR(z) liefert den Character zum ASCII-Wert z ($0 \leq z < 128$)
  - es gilt: CHR(ORD(ch)) = ch und ORD(CHR(z)) = z

- Real
  - Wertemenge: darstellbare reelle Zahlen (beschr. Stellenzahl)
  - Deklaration:

```
CONST  pi = 3.1415;
VAR  coeff,std,temp: REAL;
```

Dezimalpunkt !!

  - Ein-/Ausgabe:

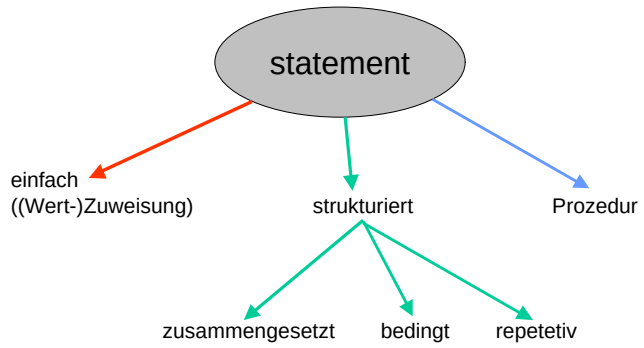
```
FROM RealInOut IMPORT   ReadReal(var),
                        WriteReal(var,minstellen);
```

  - Operationen:
    - Addition +
    - Subtraktion –
    - Multiplikation *
    - Division /
    - Vergleichsoperatoren =,<>,>=,>,<=,<
    - alle Op. auf REAL-Werten sind Approximationen (Genauigkeit von Maschine/Implementierung))
  - Konvertierung
    - FLOAT: CARDINAL / INTEGER → REAL
    - TRUNC: REAL → CARDINAL / INTEGER

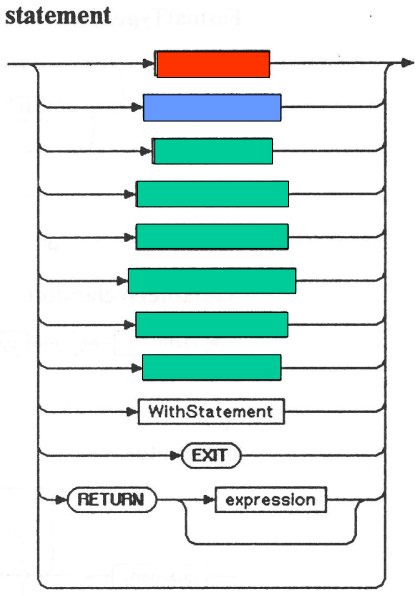
Operationsprinzipien - Anweisungen

Allgemeines

- Ein Programm manipuliert Daten
  - Operationen → Datenmanipulation (d.h. Zustandsänderung)
  - Aktionen (→ Zustandsänderung) werden in Anweisungen (statements) beschrieben



Syntaktische Definition von Anweisungen



Zuweisung und Ausdrücke

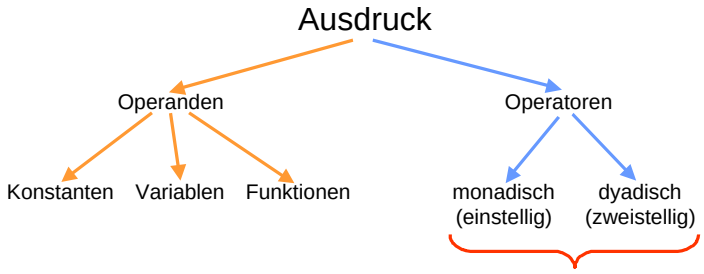
- Zuweisung („assignment“) :=
  - Syntax

  - Bedeutung, Wirkung
    - Auswertung des Ausdrucks
      → Wert des Ausdrucks muss vom Typ der Variable sein (zumindest typverträglich)
    - Zuweisung des Ergebniswertes an die Variable
      → alter Wert der Variable wird überschrieben (Zustandsänderung)
  - Beispiele
    - inkrementieren: zaehler := zaehler + 1
    - dekrementieren: zaehler := zaehler - 1
    - Werte zweier Variablen x,y tauschen

```
xalt := x
x    := y   (* x und y haben denselben Wert *)
y    := xalt
```

Zuweisung und Ausdrücke

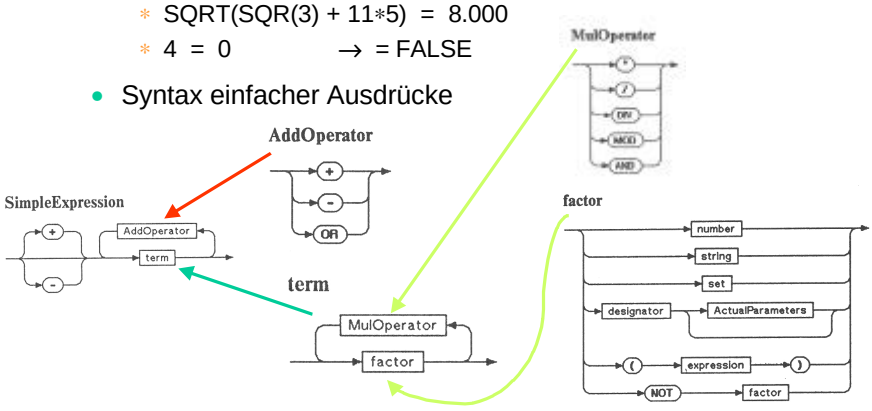
- Ausdruck („expression“)
  - Auswerteregeln: links → rechts, algebraische Regeln



Ausdrücke mit mehreren Operatoren:
- (explizit) Klammerungen
- (implizit) durch Sprachregelung festgelegte Ausführungsreihenfolge
  (z.B. „Punkt vor Strich“)

Zuweisung und Ausdrücke

- Ausdruck („expression“) (cont'd)
  - Beispiele:
    - 3.0 ∗ 8.0 / 4.0 ∗ 5.0 ⇔ ((3.0 ∗ 8.0) / 4.0) ∗ 5.0
    - 2 ∗ 3 - 4 ∗ 5 ⇔ (2 ∗ 3) - (4 ∗ 5) = -14
    - 80 / 5 / 3 ⇔ (80 / 5) / 3 = 5.333
    - SQRT(SQR(3) + 11∗5) = 8.000
    - 4 = 0 → = FALSE
  - Syntax einfacher Ausdrücke



Zuweisung und Ausdrücke

- Ausdruck („expression“) (cont'd)
  - Rangfolge von Operatoren
    - logische Negation — NOT(~)
    - Multiplikationsoperatoren — *, /, DIV, MOD, AND(&)
    - Additionsoperatoren — +, – , OR
    - Relationaloperatoren — =, <>, >=, >, <=, <, IN

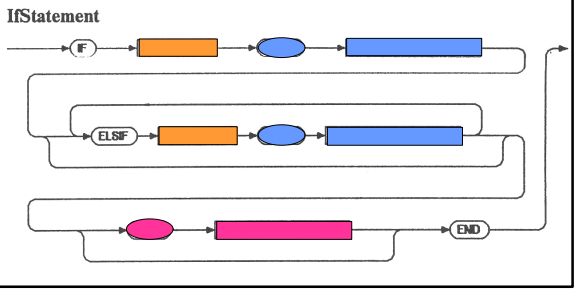
Leere Anweisungen und Anweisungsfolgen

- Leere Anweisung („empty statement“)
  - Anweisung ohne Wirkung für Programmstellen, an denen syntaktisch eine Anweisung vorkommen muss, jedoch keine Aktion stattfinden soll
- Anweisungsfolgen
  - Aneinanderreihung von Anweisungen
  - Klammerung
    - explizit: BEGIN ... END bei Moduln, Prozeduren
    - implizit: Auswahl, Wiederholung, ...

Das **;** dient lediglich der Trennung von Anweisungen, **nicht** als deren Abschluss. **;** ist **nicht** Teil der Anweisung

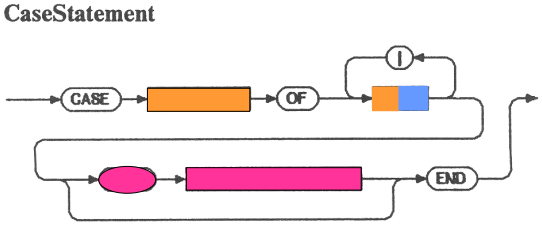
Verzweigungen - Fallunterscheidungen

- IF - Anweisung
  - Struktur: Einfachauswahl
    IF logische Bedingung THEN ... ELSE ... END
    - falls logische Bedingung = true → Abarbeitung der THEN-Anweisungsfolge ...
    - falls logische Bedingung = false → Abarbeitung der ELSE-Anweisungsfolge ...
  - Struktur: Mehrfachauswahl
    IF ... THEN ...
    ELSIF ... THEN ...
    ⋮
    ELSE ... END
  - Syntax



Verzweigungen - Fallunterscheidungen

- CASE - Anweisung
  - n unabhängige Ausgangssituationen (cases); jede ist mit einer Aktion verbunden
  - Struktur
    CASE Ausdruck zur Fallunterscheidung OF
    ... : ... ← Fall 1
    ... : ... ← Fall 2
    ⋮
    ELSE ... ← Fall n oder der „Rest“
    END
  - Syntax



Notenschlüssel

```
IF     punkte >= 100
   THEN WriteString("Note 1")
ELSIF  punkte >= 80
   THEN WriteString("Note 2")
ELSIF  punkte >= 60
   THEN WriteString("Note 3")
ELSIF  punkte >= 40
   THEN WriteString("Note 4")
ELSE    WriteString("Note 5 (nicht bestanden!)")
END;
```

```
CASE   punkte OF
   0..39 : WriteString("Note 5 (nicht bestanden!)")
  |40..59 : WriteString("Note 4")
  |60..79 : WriteString("Note 3")
  |80..99 : WriteString("Note 2")
    ELSE   WriteString("Note 1")
END;
```

Notenschlüssel

```
IF     punkte >= 100
   THEN WriteString("Note 1")
ELSIF  punkte >= 80
   THEN WriteString("Note 2")
ELSIF  punkte >= 60
   THEN WriteString("Note 3")
ELSIF  punkte >= 40
   THEN WriteString("Note 4")
ELSE    WriteString("Note 5 (nicht bestanden!)")
END;
```

```
CASE   punkte OF
   0..39 : WriteString("Note 5 (nicht bestanden!)")
  |40..59 : WriteString("Note 4")
  |60..79 : WriteString("Note 3")
  |80..99 : WriteString("Note 2")
    ELSE   WriteString("Note 1")
END;
```

Notenschlüssel

```
CASE   punkte OF
   0..39 : WriteString("Note 5 (nicht bestanden!)")
  |40..59 : WriteString("Note 4")
  |60..79 : WriteString("Note 3")
  |80..99 : WriteString("Note 2")
    ELSE   WriteString("Note 1")
END;
```

besser:

```
CASE   punkte OF
   0..39   : WriteString("Note 5 (nicht bestanden!)")
  |40..59   : WriteString("Note 4")
  |60..79   : WriteString("Note 3")
  |80..99   : WriteString("Note 2")
  |100..105 : WriteString("Note 1")
    ELSE   WriteString("unzulässige Punktezahl")
END;
```

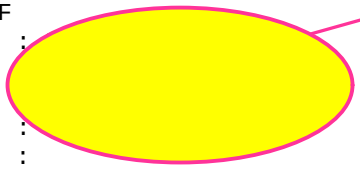
ELSE sollte (obwohl optional) in CASE nie fehlen, um unvorhergesehene Fälle aufzufangen

Auswahl

```
~
VAR
   ch  :  CHAR;
~
(* einfache Menuesteuerung zur Bibliotheksverwaltung *)
WriteString("Eingabezeichen (Kommando): ");
Read(ch);
WriteLn;

CASE   ch OF
   "S", "s" :
 | "N", "n"
 | "E", "e"
 | "A", "a" :
 | "W", "w" :                          (* leere Anweisung *)
 | "X", "x" : EXIT
    ELSE   WriteString("unzulässige Punktezahl")
END;

~
```



Wiederholungsanweisungen

- WHILE ... DO ..... END
  - vorangestellte Bedingung (kopfgesteuerte Schleife)
  - solange Bedingung gilt, führe Anweisungsfolge aus
- REPEAT ..... UNTIL ...
  - nachgestellte Bedingung (fußgesteuerte Schleife)
  - führe Anweisungsfolge aus, bis (Abbruch-)Bedingung erfüllt ist
- FOR ... TO ... [BY ...] DO ..... END
  - feste Zählschleife (kopfgesteuerte Schleife)
  - führe Anweisungsfolge aus, solange der Index sequentiell verändert wird.
- LOOP ..... END
  - einfache Schleife (loop)
  - führe Anweisungsfolge aus, solange bis Schleife per EXIT verlassen wird

Wiederholungsanweisungen

- fußgesteuerte Schleifen
  - die zu wiederholenden Anweisungen werden zunächst ausgeführt.
  - Danach wird die Schleifenbedingung geprüft
  - ist die Schleifenbedingung erfüllt, so wird an den Anfang des Anweisungsblocks gesprungen
- kopfgesteuerte Schleifen
  - bevor die zu wiederholenden Anweisungen ausgeführt werden, wird erst einmal die Schleifenbedingung geprüft.
  - ist die Schleifenbedingung nicht mehr erfüllt, so wird an das Ende des Anweisungsblocks gesprungen
  - sicherer als fußgesteuerte Schleifen