Informatik I Übung, Woche 39

Giuseppe Accaputo

24. September, 2015

Plan für heute

1. Nachbesprechung Übung 1
2. Vorbesprechung Übung 2
3. Zusammenfassung der bisherigen Vorlesungsslides

Ausdrücke

- repräsentieren Berechnungen
- bestehen aus Werten (Literale, Konstanten, Variablen), Operatoren (`+`,`-`,`...`) und Funktionen (`sin, sqrt, ...`)
- haben einen Typ und evaluieren zu einem Wert

Typumwandlungen (Typecasts)

- Implizite Typumwandlung
  - Automatisch von `INTEGER` nach `REAL`, d.h. sobald ein `REAL` (in Kombination mit `INTEGER`) in einem Ausdruck vorkommt, dann evaluiert der Ausdruck zu einem `REAL`
    - `Writeln(3.5 + 2 : 5 : 2);` → `_5.50`
- Explizite Typumwandlung
  - Umwandlung von `REAL` nach `INTEGER`
    - ```
      VAR i : INTEGER; r : REAL;
      i := Trunc(r) {Nachkommastellen
          ↪ abschneiden}
      i := Round(r) {Kaufm. Runden}
      ```
      ```

Präzedenz und Assoziativität

1. Klammern: `(...)`
2. Unäre Operatoren: `+`, `-`, `not`
3. Punkt-Operatoren: `*`, `/`, `div`, `mod`, `and`
4. Strich-Operatoren: `+`, `-`, `or`, `xor`
5. Vergleichs-Operatoren: `=`, `<>`, `<`, `>`, `<=`, `>=`

Assoziativität:

- Linksassoziativ: `A op B op C` $\Leftrightarrow$ `((A op B)op C)`
  - Punkt-, Strich- und Vergleichs-Operatoren
- Rechtsassoziativ: `A op B op C` $\Leftrightarrow$ `(A op (B op C))`
  - Unäre Operatoren

Beispiele zu Präzedenz

Finde Typ und Wert:

1. `-7 * -5 = ...`
2. `14 / 7 = ...`
3. `12 div 3 / 2 = ...`
4. `12 / 3 div 2 = ...`

Lösung

Finde Typ und Wert:

1. `-7 * -5 = 35`
   - Typ: `INTEGER`
2. `14 / 7 = 2.000000E+000`
   - Typ: `REAL`
3. `12 div 3 / 2 = 2.000000E+000`
   - Typ: `REAL`
   - Wird evaluiert als: `((12 div 3)/ 2)` (linksassoziativ)
4. `12 / 3 div 2 = ...`
   - Wird evaluiert als: `((12 / 3)div 2)` (linksassoziativ)
   - **Error:** Die Parameter von `div` sollten vom Typ `INTEGER` sein, jedoch ist der erste Parameter vom Typ `REAL`

IF-THEN-ELSE

```
IF Bedingung
THEN Anweisung;
```

- **In Worte**:
  FALLS Bedingung erfüllt ist
  DANN führe Anweisung aus

```
IF Bedingung
THEN Anweisung_1
ELSE Anweisung_2;
```

- **In Worte**:
  FALLS Bedingung erfüllt ist
  DANN führe Anweisung_1 aus
  ANSONSTEN führe Anweisung_2 aus

Verschachtelte IFs

```
IF Bedingung_1
THEN Anweisung_1
ELSE IF Bedingung_2
THEN Anweisung_2
ELSE Anweisung_3;
```

- **In Worte**:
  FALLS Bedingung_1 erfüllt ist
  DANN führe Anweisung_1 aus
  ANSONSTEN FALLS Bedingung_2 erfüllt ist
  DANN führe Anweisung_2 aus
  ANSONSTEN führe Anweisung_3 aus
  (Bedingung_1 und Bedingung_2 sind nicht erfüllt)

Blöcke

- Sequenz von Anweisungen (zwischen `BEGIN` und `END`)
- Wird wie eine einzelne Anweisung verwendet

```
BEGIN
    preis := 12.10;
    k := 0;
END
```

$\Longrightarrow$ Block bestehend aus mehreren Anweisungen, wird jedoch als eine einzelne Anweisung interpretiert

```
preis := 12.10;
k := 0;
```

$\Longrightarrow$ zwei Anweisungen

Der Datentyp BOOLEAN

```
VAR
    a : BOOLEAN;
```

- mögliche Werte: `true` und `false`
- Vergleichsoperatoren: `=`, `<>`, `<`, `>`, `<=`, `>=`
- Weitere Operatoren: `and`, `or`, `xor`, `not`

Zu beachten bei IFs (1/2)

```
IF Bedingung
THEN Anweisung;
```

- Bedingung muss vom Typ BOOLEAN sein
  - Verwende nur BOOLEAN Variablen (z.B. VAR a : BOOLEAN;) oder Operatoren, welche ein Ergebnis vom Typ BOOLEAN zurückgeben (z.B. not, or, and, <, >, =, etc.)
- Anweisung (inkl. Anweisung_* bei IF-THEN-ELSE und verschachtelten IFs) kann eine einzelne Anweisung sein oder ein Block bestehend aus mehreren Anweisungen
- Platzierung des Semikolons ;

Zu beachten bei IFs (2/2)

program_1.pas (Ausschnitt):

```
IF a = 10 THEN
    b := 12;
ELSE
    b := 13
```

program_2.pas (Ausschnitt):

```
IF a = 10 THEN
    b := 12;
    c := 14
ELSE
    b := 13;
```

Boole'sche Operatoren

- `and`, `or`, `xor`, `not`
- Präzedenz:
  1. `not`
  2. `and`
  3. `or`, `xor`
- Funktionssignaturen:
  - `and`, `or`, `xor`:
    $\texttt{BOOLEAN} \times \texttt{BOOLEAN} \to \texttt{BOOLEAN}$
  - `not`:
    $\texttt{BOOLEAN} \to \texttt{BOOLEAN}$

and

`a and b`

- Wenn `a` **und** `b` beide den Wert `true` haben, dann evaluiert `a and b` zu `true`, ansonsten zu `false`

a	b	a and b
true	true	true
true	false	false
false	true	false
false	false	false

`or`

`a or b`

- Wenn `a` **oder** `b` (oder beide gleichzeitig) den Wert `true` haben, dann evaluiert `a or b` zu `true`, ansonsten zu `false`

a	b	a or b
true	true	true
true	false	true
false	true	true
false	false	false

xor

`a xor b`

- Wenn `a` und `b` unterschiedliche Werte haben (z.B. `a = true;` und `b = false;`), dann evaluiert `a xor b` zu `true`, ansonsten zu `false`

a	b	a xor b
true	true	false
true	false	true
false	true	true
false	false	false

`not`

`not a`

- Aus `true` wird `false` und aus `false` wird `true` (Negierung)

a	not a
true	false
false	true

Vergleichsoperatoren

Ordnung

- `INTEGER`, `REAL`: entsprechend numerischem Wert
  - $2.0 < 2.3$
- `STRING`: Lexikographisch
  - Sonderzeichen < Grossbuchstaben < Kleinbuchstaben
    - `'@Hello'`< `'world!'` ⇔ `true`
    - `'world!'`< `'Hello'` ⇔ `false`
    - `'world␣cup'`< `'world␣domination'` ⇔ `true`
- `BOOLEAN`: `false` < `true`