Informatik I Übung, Woche 41

Giuseppe Accaputo

9. Oktober, 2014

Plan für heute

1. Fragen & Nachbesprechung Übung 3
2. Zusammenfassung der bisherigen Vorlesungsslides
3. Tipps zur Übung 4

Aufgabe 3.1 (1/2)

- Unterschied zwischen := und =
  - := ist der Zuweisungsoperator

```
a := 3;
```

  - = ist der Vergleichsoperator

```
a,b : BOOLEAN;
...
IF a = b THEN
```

Aufgabe 3.1 (2/2)

- Präzedenz der Boole'schen Operatoren
  - Gegeben:

```
a = true; b = 1; c = 2;
IF b > c or a THEN ...
```

$\Longrightarrow$ Falsch: Compiler interpretiert die `IF`-Abfrage als `IF b > (c or a)` und gibt einen Fehler aus, da `or` zwei `BOOLEAN` Parameter verlangt, jedoch der Parameter `c` vom Typ `INTEGER` ist

$\Longrightarrow$ Korrekt:

```
IF (b > c) or a THEN ...
```

  - **Tipp**: Verwende Klammern sobald Vergleichsoperatoren, Boole'sche Operatoren, Zahlen (`REAL`, `INTEGER`, etc.) und `BOOLEAN` Variablen in einer Bedingung vorkommen

Aufgabe 3.2

```
IF (a < b) THEN Anweisung_1
ELSE IF (c > d) THEN Anweisung_2
ELSE IF (e = f) THEN Anweisung_1
...
```

$\Longrightarrow$ Code kann vereinfacht werden, indem die Bedingungen der IF-ELSE-IF-Zweige, welche das genau Gleiche tun (hier beispielsweise die beiden Zweige, welche `Anweisung_1` ausführen) zusammen *verodert* (`or`) werden:

```
IF (a < b) or (e = f) THEN Anweisung_1
ELSE IF (c > d) THEN Anweisung_2;
...
```

Bits und Bytes

- Zahlen werden im Computer als n-stellige Binärzahlen dargestellt
- 1 Bit kann Wert 0 oder 1 annehmen
- 1 Byte besteht aus 8 Bits
- 1 Word beschreibt die Grösse einer Speicherzelle ($8, 16, 32, 64$ Bits)

Darstellung von Dezimalzahlen im Binärsystem

Beispiel: Dezimalzahlen als 8-stellige Binärzahlen darstellen:

Wertigkeit	128	64	32	16	8	4	2	1
94 binär $=$	0	1	0	1	1	1	1	0

$\Longrightarrow \; 94 = 64 + 16 + 8 + 4 + 2$

- 8-stellige Binärzahl $\Longrightarrow$ der grösste Wert in der Binärzahl ist $2^7 = 128$
  - 7. Stelle der Binärzahl repräsentiert 2^7
    6. Stelle der Binärzahl repräsentiert 2^6
    $\ldots$
    0. Stelle der Binärzahl repräsentiert $2^0 = 1$

Darstellung von negativen Dezimalzahlen im Binärsystem

- Zweierkomplement verwenden, um negative Zahlen im Binärsystem darzustellen
  - $-x$ wird so dargestellt, dass $-x + x = 0$
- Vorgehen:
  1. Zahl ohne Vorzeichen ins Binärsystem umrechnen
  2. alle Bits flippen ($0 \to 1$ und $1 \to 0$)
  3. 1 addieren
- Beispiel: -4
  1. $4 = 00000100$
  2. Alle Bits flippen: `not` $00000100 = 11111011$
  3. 1 addieren: $11111011 + 00000001 = 11111100$
  4. Kontrolle:
     $11111100 = -128 + 64 + 32 + 16 + 8 + 4 = -4$ ✓
     oder $11111100 + 00000100 = 0$ ✓

Repräsentation von Dezimalzahlen

- Zwei Arten von Datentypen um Dezimalzahlen darzustellen: Unsigned und Signed
  - **Unsigned**: nur positive Werte werden dargestellt
  - **Signed**: positive und negative Werte können dargestellt werden
    - Binärzahlen, die eine Dezimalzahl durch einen *signed* Datentypen darstellen und als erste Ziffer eine 1 haben sind negativ
- Beispiel: 11010101
  - Unsigned:
    $11010101 = 128 + 64 + 16 + 4 + 1 = 213$
  - Signed:
    $11010101 = -128 + 64 + 16 + 4 + 1 = -43$

Überläufe (Overflows) (1/3)

```
VAR a : INTEGER;

BEGIN
    a := 39364;
    Writeln(a);
END.
```

$\Longrightarrow$ Ausgabe auf der Konsole ist -26172. Warum?

Überläufe (Overflows) (2/3)

Problem: Free Pascal verwendet für `INTEGER` automatisch `SMALLINT`, jedoch ist der Wertebereich von `SMALLINT` zu klein um 39364 korrekt darzustellen (Wertebereich: $[-32768, 32767]$)

- 39364 als `SMALLINT` (16-stellige Binärzahl, 16 Bits):
  1001100111000100 ✗
  $\Longrightarrow$ 39364 als `SMALLINT` resultiert in -26172

- 39364 als `LONGINT` (32-stellige Binärzahl, 32 Bits):
  $00000000000000001001100111000100$ ✓

Überläufe (Overflows) (3/3)

Verwende von nun an in jedem Programm folgenden Code-Block:

```
PROGRAM ...

{$OverflowChecks ON}
{$RangeChecks ON}
TYPE INTEGER = LONGINT;

VAR ...
```

- Compiler überprüft nun, ob Überläufe entstehen und gibt eine Fehlermeldung aus
- `INTEGER` ist nun immer ein 32-Bit `LONGINT`

DOs and DON'Ts: Addiere zuerst kleine Zahlen

$$10^{16} + 1 + \cdots + 1 \neq 1 + \ldots 1 + 10^{16}$$

- Addiert man zuerst die grossen Zahlen und dann die kleinen Zahlen, so rationalisiert der Compiler dies so, dass die kleine Zahlen keinen signifikanten Beitrag leisten und ignoriert diese