

Informatik I Übung, Woche 40

Giuseppe Accaputo

2. Oktober, 2014

Plan für heute

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

IF-Abfrage

- ▶ Eine **IF**-Abfrage ist eine Anweisung; deshalb wird sie mit einem Strich-Punkt abgeschlossen sollte eine weitere Anweisung darauf folgen

```
IF ... THEN Anweisung;
```

```
IF ... THEN Anweisung_1  
ELSE Anweisung_2;
```

```
IF ... THEN Anweisung_1  
ELSE IF ... THEN Anweisung_2  
ELSE Anweisung_3;
```

Syntaxfehler

- ▶ **Syntaxfehler:** Fehler im Programmcode, die durch unkorrektes Verwenden der Sprache erzeugt und vom Compiler während der Kompilierung erkannt werden
 - ▶ Beispiele: fehlender Strich-Punkt zwischen zwei Anweisungen, ungültiger Variablennamen
- ▶ **Analogie:** Artikel für ein Magazin schreiben
 - ▶ Author des Artikels \Leftrightarrow Programmierer
 - ▶ Artikel \Leftrightarrow Programmcode
 - ▶ Person, welche beim Magazin für die Veröffentlichung des Textes verantwortlich ist und den Text vor der Veröffentlichung auf grammatikalische Fehler überprüft \Leftrightarrow Compiler
 - ▶ Grammatikalische Fehler im Artikel \Leftrightarrow Syntaxfehler

Welche Fehler erkennt der Compiler nicht?

► **Semantische Fehler:** Fehler in der Bedeutung des Programms

- Beispiel: Wir führen eine *Anweisung* nur dann aus, wenn der Benutzer über 18 Jahre alt ist, schreiben jedoch im Programmcode

```
IF alter < 18 THEN Anweisung;
```

Der Compiler wird diesen Code kompilieren da er syntaktisch korrekt ist, jedoch ist er semantisch falsch (*Anweisung* wird für Benutzer ausgeführt, welche unter 18 Jahre alt sind).

Der in dieser Situation semantisch korrekte Code würde wie folgt aussehen:

```
IF alter > 18 THEN Anweisung;
```

Aufgabe 2.1: Prog5.pas

Welches sind die Syntaxfehler (die Fehler, welche vom Compiler erkannt werden) im folgenden Codeausschnitt?

```
...  
  IF y < 0 THEN  
    x := 1; z := 3;  
  
  IF y = 0 THEN x:=5;  
  ELSE x := 6;  
...
```

Aufgabe 2.1: Prog5.pas und Semantik

```
...  
  IF y < 0 THEN  
    BEGIN  
      x := 1;  
      z := 3;  
    END  
  
  IF y = 0 THEN x := 5  
  ELSE x := 6;  
...  

```

De Morgansche Regeln

1. `not (a and b) ≡ (not a) or (not b)`

2. `not (a or b) ≡ (not a) and (not b)`

Short-Circuit Auswertung (Kurzschlussauswertung)

Regeln:

- ▶ a **and** b: Werte zuerst a aus; werte b nur aus falls a wahr ist
- ▶ a **or** b: Werte zuerst a aus; werte b nur aus falls a falsch ist

Beispiele:

- ▶ Folgende Werte sind gegeben:
a := 1; b := 2; c := **false**;

Was wird in den folgenden Zeilen ausgewertet?

- ▶ **IF** (a = b) **and** c **THEN** Anweisung_1;
- ▶ **IF** (a = b) **or** c **THEN** Anweisung_2;

WHILE-DO

```
WHILE Bedingung DO Anweisung;
```

▶ **In Worte:**

SOLANGE Bedingung wahr ist, führe Anweisung aus

- ▶ Programmierer ist verantwortlich dafür, dass die **WHILE**-Schleife nach einer endlichen Anzahl Iterationen abbricht

WHILE-DO ein: Beispiel

```
VAR
    a : BOOLEAN;
BEGIN
    a := true;
    WHILE a DO Writeln('Hello, □world!')
END.
```

- ▶ Wie sieht die Ausgabe auf der Konsole aus?

FOR-Schleife: Syntax

1.

```
FOR Variable := Startwert TO Endwert  
DO Anweisung;
```

2.

```
FOR Variable := Startwert DOWNTO Endwert  
DO Anweisung;
```

► **In Worte:**

Zähle Variable von Startwert bis Endwert und führe jedes mal Anweisung aus

FOR-Schleife: Informationen zur Anwendung

- ▶ Variable, Startwert und Endwert müssen als **INTEGER** Variablen deklariert werden:

```
VAR
```

```
Variable, Startwert, Endwert :  
    ↪ INTEGER;
```

- ▶ Anweisung kann eine einzelne Anweisung sein oder ein Block bestehend aus mehreren Anweisungen

FOR-Schleife: Informationen zur Anwendung

- ▶ Variable ist eine *read-only* Variable, d.h. sie kann in der FOR-Schleife nicht verändert werden (nur lesen erlaubt).

Folgendes funktioniert also nicht:

```
FOR Variable := Startwert TO Endwert DO  
  ↪ Variable := Variable + 1;
```

⇒ Variable wird verändert: Variable := Variable + 1

Folgendes funktioniert stattdessen:

```
FOR Variable := Startwert TO Endwert DO  
  ↪ NeueVariable := Variable + 1;
```

⇒ Variable wird nur gelesen (read-only)

FOR-Schleife: ein Beispiel

```
VAR
    start_v, end_v, curr_v : INTEGER;
BEGIN
    start_v := 1;
    end_v := 5;

    FOR curr_v := start_v TO end_v
    DO Writeln(curr_v);
END.
```

- ▶ Wie sieht die Ausgabe auf der Konsole aus?

REPEAT-UNTIL

```
REPEAT Anweisung_1; Anweisung_2; ...;  
    ↪ Anweisung_N; UNTIL Bedingung;
```

► **In Worte:**

Wiederhole die Anweisungssequenz

Anweisung_1, Anweisung_2, ..., Anweisung_N bis
Bedingung erfüllt ist

REPEAT-UNTIL: Beispiel

Fordere den Benutzer auf, für das Geschlecht entweder `m` oder `w` einzugeben. Gibt der Benutzer stattdessen einen anderen Buchstaben ein, so wird er solange aufgefordert das Geschlecht einzugeben, bis er korrekterweise entweder `m` oder `w` eingegeben hat:

```
REPEAT
    Write('Bitte  Geschlecht  eingeben  (m/w): 
        ↪ ');
    Readln(input);
UNTIL (input = 'm') or (input = 'w');
```

Wann soll ich welche Schleife verwenden?

- ▶ Verwende eine **FOR**-Schleife wenn die Anzahl der Iterationen *bekannt* ist
- ▶ Verwende eine **WHILE-DO**-Schleife wenn die Anzahl der Iterationen *unbekannt* ist
- ▶ Unterschied zwischen **WHILE-DO** und **REPEAT-UNTIL**:
 - ▶ **WHILE-DO** überprüft zuerst ob Bedingung erfüllt ist und führt erst dann die Anweisung aus
 - ▶ **WHILE-DO**-Schleife wird abgebrochen, wenn Bedingung **nicht erfüllt** ist
 - ▶ **REPEAT-UNTIL** führt die Anweisung mindestens einmal aus und überprüft erst dann, ob die Bedingung erfüllt ist
 - ▶ **REPEAT-UNTIL**-Schleife wird abgebrochen, wenn Bedingung **erfüllt** ist

FOR-TO → WHILE-DO

Jede **FOR-TO**-Schleife kann in eine **WHILE-DO**-Schleife umgewandelt werden (Umkehrung gilt nicht immer)

```
FOR Variable := Startwert TO Endwert  
DO Anweisung;
```

ist äquivalent zu

```
Variable := Startwert;  
  
WHILE Variable <= Endwert DO  
  BEGIN  
    Anweisung;  
    Variable := Variable + 1;  
  END
```

Tipps zur Übung 3

- ▶ Aufgabe 3.1: Unbedingt lösen!
- ▶ Aufgabe 3.2: Auf Programmierstil achten

```
▶ student := str = 'w';
```

ist die elegantere Version des folgenden Codeausschnitts (beide tun dasselbe):

```
IF str = 'w' THEN  
    student = true  
ELSE  
    student = false;
```

- ▶ Aufgabe 3.3: Bezüglich der Wahl der Schliefe siehe Slide 18

Reminder: Präzedenz und die Boole'schen Operatoren

Präzedenz:

- ▶ Boole'sche Operatoren haben höhere Präzedenz als Vergleichsoperatoren:
 1. Klammern: (...)
 2. Unäre Operatoren: +, -, `not`
 3. Punkt-Operatoren: *, /, `div`, `mod`, `and`
 4. Strich-Operatoren: +, -, `or`, `xor`
 5. Vergleichs-Operatoren: =, <>, <, >, <=, >=

Beispiel: Präzedenz und die Boole'schen Operatoren

Beispiel:

```
VAR
    a, b, c, d : INTEGER;
BEGIN
    a := 1; b := 2; c := 3; d := 4;
    IF a < b and c < d THEN
        Writeln('Yes')
    ELSE
        Writeln('No');
END.
```

- ▶ Wie sieht die Ausgabe auf der Konsole aus?

Beispiel: Präzedenz und die Boole'schen Operatoren

Beispiel:

```
VAR
    a, b, c, d : INTEGER;
BEGIN
    a := 1; b := 2; c := 3; d := 4;
    IF (a < b) and (c < d) THEN
        Writeln('Yes')
    ELSE
        Writeln('No');
END.
```

⇒ Wird nun erfolgreich kompiliert und gibt Yes aus