Informatik I Übung, Woche 45

Giuseppe Accaputo

6. November, 2014

Plan für heute

1. Lernziele für die heutige Übungsstunde
2. Nachbesprechung Übung 7
3. Vorbesprechung Übung 8

Lernziele

- Tipps zur Fehlerbehebung
- Initialisierung von Variablen
- `RECORD` (Datensatz)

Tipps zur Fehlerbehebung

- **Semantische Fehler**: Damit ich semantische Fehler anschauen kann, benötige ich von euch die genaue Eingabe (oder Eingabereihenfolge), welche zum Fehler führt.
- **Syntaktische Fehler**: Installiert einen Text-Editor, welcher es erlaubt gezielt auf Zeilen zu springen mit der Angabe der Zeilennummer (Fehlermeldungen zu Syntaxfehlern beinhalten immer eine Zeilennummer; in der Umgebung dieser Zeile kann sich der angegebene Fehler befinden, manchmal auch genau auf dieser Zeile)
  - Mac OS X: TextMate, `APFEL + L`
  - Windows, Linux: SublimeText, `CTRL + G`

Initialisierung von Variablen

Tipp: Initialisiert alle Variablen welche
ihr in eurem Code verwendet, inkl. der Rückgabewert einer Funktion.

Zitat aus der offiziellen Free Pascal Dokumentation:

"By default, variables in Pascal are not initialized after their declaration. Any assumption that they contain 0 or any other default value is erroneous: They can contain rubbish ." (`http://www.freepascal.org/docs-html/ref/refse23.html`)

Initialisierung von Variablen und Rückgabewerten

Initialisierung von Variablen:

```
VAR a : INTEGER; b : BOOLEAN; { Deklaration
    ↪ }
...
a := 0; { Beispiel-Initialisierung }
b := false;
```

Initialisierung des Rückgabewerts:

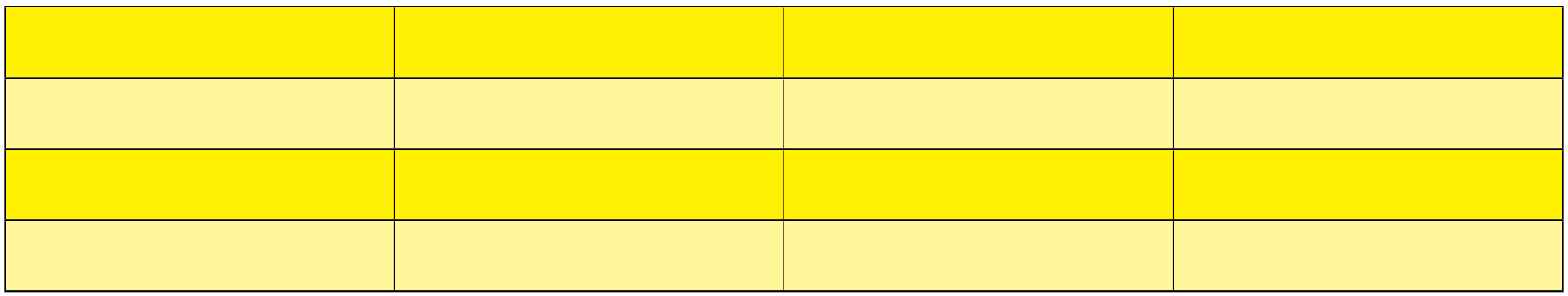
```
FUNCTION  HasWon (...): BOOLEAN;
...
HasWon := false; { Initialisierung }

IF (...) THEN HasWon := true; END;
```

Multidimensionale Arrays

Zeilenweise durchiterieren: (`i` = Zeile, `j` = Spalte)

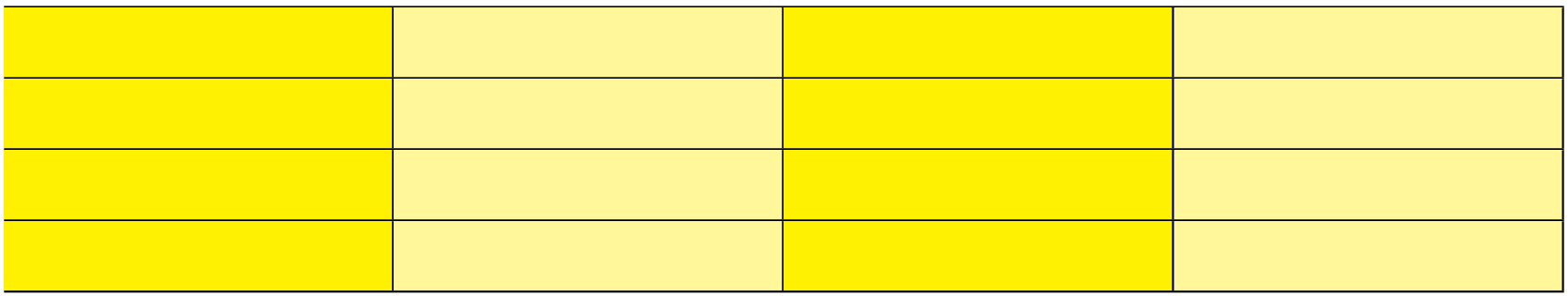
```
FOR i:= 1 TO size DO
  FOR j:= 1 TO size DO
    c[i,j] := ...
```



Multidimensionale Arrays

Spaltenweise durchiterieren: (`i` = Zeile, `j` = Spalte)

```
FOR j:= 1 TO size DO
  FOR i:= 1 TO size DO
    c[i,j] := ...
```



RECORD (Datensatz): Syntax

```
RECORD
    Var1 : REAL;
    Var2 : INTEGER;
    Var3 : STRING;
    Var4 : REAL;
    ...
END
```

Benutzerdefinierte Typen

Wir können benutzerdefinierte Typen definieren (Typen, die einen von uns gegebenen Namen haben):

```
TYPE MeinTypNamen = INTEGER;
```

RECORD (Datensatz): Semantik

- Speichert mehrere Variablen von beliebigen Typen ab
- Geeignet um verschiedene Eigenschaften eines Objektes zusammenzufassen, z.B. Auto (siehe Codeausschnitt unten)
- Kann auch als Typ verwendet werden

```
      { Recordname }
TYPE TAuto = RECORD
                  { Feldnamen }
                  marke, farbe : STRING;
                  preis : REAL;
             END
```

RECORD (Datensatz): Zugriff

Um auf Feldnamen eines Records zuzugreiffen kann man . verwenden:

```
VAR auto : TAuto;
...
{ Schreibzugriff }
auto.preis := 19000;
auto.marke := 'Volkswagen';
{ Lesezugriff }
IF auto.farbe = 'Gelb' THEN ...
```

Array von RECORDs

Wir definieren ein Array bestehend aus 3 Autos:

```
autos : ARRAY [1..3] OF TAuto; {Deklaration}
```

Nach der Deklaration befinden sich im Array nun 3 Autos (`TAuto`) die weder eine Marke, eine Farbe noch einen Preis haben:

```
autos =
```

1	2	3
TAuto marke := ? farbe := ? preis := ?	TAuto marke := ? farbe := ? preis := ?	TAuto marke := ? farbe := ? preis := ?

Array von RECORDs: Initialisierung der Datensätze (1/4)

Wir müssen nun die einzelnen Autos mit einer Marke, Farbe und einem Preis versehen.

Wir können nun beispielsweise das erste Auto im Array initialisieren. Dazu greifen wir zuerst auf das erste Element im Array zu mittels `autos[1]` und verwenden dann den ".-Operator" um die Felder für die Marke, Farbe und den Preis mit Werten zu versehen:

```
autos[1].marke := 'Audi';
autos[1].farbe := 'Gelb';
autos[1].preis := 30000;
```

Array von RECORDs: Initialisierung der Datensätze (2/4)

Nachdem wir also das erste Auto initialisiert haben sieht unser Array wie folgt aus:

```
autos =
```

1	2	3
TAuto marke := 'Audi' farbe := 'Gelb' preis := 30000	TAuto marke := ? farbe := ? preis := ?	TAuto marke := ? farbe := ? preis := ?

Array von RECORDs: Initialisierung der Datensätze (3/4)

Wir können nun die beiden restlichen Autos initialisieren:

```
{ Zweites Auto initialisieren }
autos[2].marke := 'VW';
autos[2].farbe := 'Rot';
autos[2].preis := 20000;
{ Drittes Auto initialisieren }
autos[3].marke := 'Mini';
autos[3].farbe := 'Blau';
autos[3].preis := 25000;
```

Array von RECORDs: Initialisierung der Datensätze (4/4)

Nach der Initialisierung aller Autos hat unser Array folgende Form:

```
autos =
```

1	2	3
TAuto marke := 'Audi' farbe := 'Gelb' preis := 30000	TAuto marke := 'VW' farbe := 'Rot' preis := 20000	TAuto marke := 'Mini' farbe := 'Blau' preis := 25000

Array von RECORDs: Auf Datensätze zugreifen (1/2)

Nun können wir auf die Autos zugreifen und mit diesen arbeiten. Dazu verwendet man wie bis anhin mit Arrayzugriffen `autos[i]` um auf das (in unserem Beispiel) i-te Auto im Array zuzugreifen. Anschliessend verwendet man den ".-Operator" um auf die einzelnen Felder des Autos zuzugreifen:

```
IF autos[1].name = 'Audi' THEN ...
```

Array von RECORDs: Auf Datensätze zugreifen (2/2)

An der Art wie wir mittels eines FOR-Schleife durch das Array iterieren ändert sich auch nichts (LOW,HIGH können weiterhin verwendet werden), da wir stehts mit einem Array zu tun haben, nur das im Unterschied zu den vorherigen Übungen statt INTEGERs oder REALs das Array TAutos enthält:

```
FOR i := LOW(autos) TO HIGH(autos) DO
   IF autos[i].preis > 10000 THEN ...
```

Merke: autos[i] ist nun ein TAuto. Mit autos[i] direkt können wir nichts anfangen, jedoch können wir mit dessen Felder arbeiten. Dazu müssen wir den ".-Operator" verwenden um Zugriff auf die Felder des Datensatzes autos[i] zu erhalten, z.B. autos[i].preis.

Aufgabe 8.1

- Nur einige Anpassungen vornehmen
- Zugriff auf `RECORD` mit `.`