Informatik I Übung, Woche 43

Giuseppe Accaputo

23. Oktober, 2014

Plan für heute

1. Lernziele für die heutige Übungsstunde
2. Nachbesprechung Übung 5
3. Vorbesprechung Übung 6

Lernziele

- Normalisierung von Gleitkommazahlen
- `FUNCTION` und `PROCEDURE`
- Wert- und Referenzparameter
- Open Arrays
- Vor- und Nachbedingungen

Aufgabe 5.1: Währungsrechner

Tipp für die Prüfung: implementiert genau das, was in der Aufgabe steht, d.h. dieselben Eingaben ermöglichen, dieselben Textpassagen ausgeben, etc.

Quiz-Frage 1

Frage: Können negative Zahlen mit dem `CARDINAL` Datentyp dargestellt werden?

Quiz-Frage 1

Frage: Können negative Zahlen mit dem `CARDINAL` Datentyp dargestellt werden?

Antwort: Nein. `CARDINAL` ist ein vorzeichenloser (*unsigned*) Datentyp (Wertebereich: $[0,\ 4294967295]$); negative Zahlen können nicht dargestellt werden mit dem `CARDINAL` Datentyp

Aufgabe 5.2: Normalisierung von Gleitkommazahlen

Berechnung von $957.8 + 50.37$ (4 Stellen für die Mantisse und 2 Stellen für den Exponenten):

```
     9  5  7  8  |  0  2
 +   0  5  0  3  |  0  2
=================
1  |  0  0  8  1  |  0  2
```

Bei der Normalisierung sollte nicht vorne abgeschnitten werden, sondern der Exponent so erhöht werden, dass die Zahl wieder in die Mantisse passt da ansonsten ein zu grosser Wertverlust entsteht:

$$1\ |\ 0\ 0\ 8\ 1\ |\ 0\ 2 \overset{\text{normalisieren}}{\Longrightarrow} 1\ 0\ 0\ 8\ |\ 0\ 3 = 1.008 \cdot 10^3$$

Der Rundungsfehler beträgt also $1008.17 - 1008 = \underline{0.17}$

Funktionen und Prozeduren: Warum?

- Code-Duplikation vermeiden
  - Mehrfach vorkommende Code-Ausschnitte können in eine Funktion/Prozedur zusammengefasst werden
  - Änderungen müssen nur noch zentral bei der Funktion/Prozedur vorgenommen werden
- Lesbarkeit des Codes verbessern
  - Andere Programmieren werden euch dankbar sein

`FUNCTION`: Definition

Eine `FUNCTION`...

- ...hat einen Namen
  z.B. `isPrime`
- ...nimmt $i = 0, \ldots, n$ Variablen entgegen
  z.B. `(a : INTEGER; b : REAL)`
- ...gibt ein Resultat zurück, wessen Typ durch den Rückgabetyp definiert ist

Syntax einer FUNCTION

```
FUNCTION Fkt (Var1 : Typ1; ...) : RTyp;
VAR
    i : Typ2;
    ret : RTyp;
BEGIN
    i := ...;
    ret := ...;
    Fkt := ret;
END;
```

Syntax einer FUNCTION: Der Kopf

```
FUNCTION Fkt (Var1 : Typ1; ...) : RTyp;
```

- Kopf der Funktion
- `Fkt` ist der Name der Funktion
- `(Var1 : Typ1; ...)` ist die Liste der Parameter, welche die Funktion entgegennimmt.
  - `Typ1` kann `INTEGER`, `REAL`, etc. sein
  - Kann beliebig viele Variablen enthalten, z.B. `(Var1 : Typ1; ...; VarN : TypN;)`
- `RTyp` ist der Rückgabetyp der Funktion, z.B. `INTEGER`, `REAL`, etc.
- Aufruf: `res := Fkt(Wert1, ...)`
  - Wichtig: Typen müssen passen (z.B. `Wert1` muss vom Typ `Typ1` sein)

Syntax einer FUNCTION: Die Deklarationen

```
VAR
    i : Typ2;
    ret : RTyp;
```

- Deklarationen: Hier werden Variablen deklariert, welche nur in der Funktion `Fkt` verwendet werden können (lokale Variablen, auch Hilfsvariablen genannt)

Syntax einer FUNCTION: Der Rumpf

```
BEGIN
    i := ...;
    ret := ...;
    Fkt := ret;
END;
```

- Hier können die lokalen Variablen, welche im Deklarationen-Teil definiert wurden verwendet werden
- `Fkt := ret;` muss immer angegeben werden, da dies das Resultat der Funktion ist welches zurückgegeben wird

PROCEDURE: Definition

Eine PROCEDURE ist eine FUNCTION keinen Rückgabewert hat, also auch nichts zurückgibt.

PROCEDURE: Syntax

```
PROCEDURE Proc (Var1 : Typ1; ...);
VAR
    i : Typ2;
    j : Typ3;
BEGIN
    i := ...;
    j := ...;
    ...
END;
```

Quiz-Frage 2

```
FUNCTION IsPrime (n : INTEGER) : BOOLEAN;
VAR i : INTEGER;
BEGIN
    i := 2;
    WHILE n mod i <> 0 DO Inc(i);
    IsPrime := i = n;
END;
```

Frage:

- Funktionsname?
- Anzahl Variablen?
- Rückgabetyp?

Quiz-Frage 2

```
FUNCTION IsPrime (n : INTEGER) : BOOLEAN;
VAR i : INTEGER;
BEGIN
    i := 2;
    WHILE n mod i <> 0 DO Inc(i);
    IsPrime := i = n;
END;
```

Antwort:

- Funktionsname: `IsPrime`
- Anzahl Variablen: 1
- Rückgabetyp: `BOOLEAN`

Teamarbeit: Die `myAND`-Funktion definieren

Aufgabe: Definiert die Funktion `myAND`, welche zwei `BOOLEAN`-Variablen entgegennimmt und `TRUE` zurückgibt, falls beide `BOOLEAN`-Variablen `TRUE` sind und `FALSE` in allen anderen Fällen.

Beispiel-Implementation der myAND-Funktion

```
FUNCTION myAND (a : BOOLEAN; b : BOOLEAN) :
    ↪ BOOLEAN;
BEGIN
    myAND := a and b;
END
```

Wertparameter vs. Referenzparameter

Wertparameter	**Referenzparameter**
`PROCEDURE P (i : REAL)`	`PROCEDURE P (VAR i : ↪ REAL)`
`P(n)`: `n` wird ausgewertet und übergeben	`P(n)`: `n` wird als Referenz übergeben
`i` ist eine lokale Variable	`i` ist ein Alias für die Variable `n` (neuer Name für die gleiche Variable)
Darf bei `FUNCTION` und `PROCEDURE` verwendet werden	Darf nur bei `PROCEDURE` verwendet werden

Analogie zu Wertparameter und Referenzparameter

Ich bin eine Funktion, welche ein leeres Blatt Papier als Parameter entgegen nimmt. Ihr übergibt mir nun ein leeres Blatt Papier.

Fall **Wertparameter**:

- Ich kopiere das leere Blatt Papier und mache eigene Notizen darauf. Nach meiner Ausführung verwerfe ich das Blatt Papier mit den Notizen
- Ihr seid immer noch im Besitz des leeren Blatt Papier

Fall **Referenzparameter**:

- Ich schreibe direkt auf euer beschriftetes Blatt Papier meine Notizen
- Nach meiner Ausführung seid ihr neu im Besitz eines Blatt Papiers, das meine Notizen enthält

Warum Referenzparameter verwenden?

Stellt euch vor, ihr möchtet gerne mehrere Resultate zurückgeben.

Problem: `FUNCTION` erlaubt es, nur einen Wert zurückzugeben.

Lösung: Verwende eine Prozedur mit Referenzparametern, wobei die Referenzparameter verwendet werden können, um die verschiedenen Resultate zu speichern:

```
PROCEDURE P (VAR res1, res2, res3 : INTEGER)
    ↪ ;
BEGIN
    res1 := 1+2;
    res2 := 4 div 2;
    res3 := 4*5;
END
```

Quiz-Frage 3

Frage: Was ist der Wert von `i` nach dem Aufruf von `MyF(i)`?

```
PROCEDURE MyF (VAR i : INTEGER);
BEGIN
    i := 3;
END

...
BEGIN
    i := 10;

    MyF(i);
```

Quiz-Frage 3

Frage: Was ist der Wert von `i` nach dem Aufruf von `MyF(i)`?

```
PROCEDURE MyF (VAR i : INTEGER);
BEGIN
    i := 3;
END

...
BEGIN
    i := 10;

    MyF(i);
```

Antwort: `i` hat den Wert 3 nach dem Aufruf von `MyF(i)`. Grund dafür ist, dass `MyF` einen Referenzparameter entgegen nimmt.

Arrays als Funktions- und Prozedur-Argumente

Open Array:

- Arrays können als Funktions- und Prozedur-Argumente verwendet werden
- Array *ohne Grössenangabe*
- Statisches und Dynamisches Array als Argument möglich
  - Wichtig: Auf Open Arrays kann `SETLENGTH` nicht angewendet werden!
- Indizes sind immer von 0 bis $n-1$
  - Wichtig: Vorallem zu beachten, wenn statisches Array übergeben wird da beliebige Indexangabe möglich ist, z.B. `ARRAY [1000...4582] OF ...`

Arrays als Funktions- und Prozedur-Argumente

Syntax: `FUNCTION M (wer : ARRAY OF INTEGER)...`

- Statisches oder dynamisches Array übergeben:

```
M(arr)
```

- Array bestehend aus den Elementen $\text{arr}_3, \text{arr}_4, \text{arr}_5$ übergeben (Slice):

```
M(arr[3..5])
```

Vor- und Nachbedingungen

Es muss klar sein, was eine Funktion / Prozedur erreicht. Dies können wir mit Kontrakte erreichen, welche durch eine Vor- und Nachbedingung definiert werden.

- **Vorbedingung** (engl. *Precondition*): Was muss erfüllt sein
  - Ist Vorbedingung nicht erfüllt, können wir auf unerwartetes Verhalten stossen
  - Vorbedingungen können mit `ASSERT` überprüft werden
- **Nachbedingung** (engl. *Postcondition*): Was wird erreicht

Tipps zur Übung 6

Aufgabe 6.1:

- Relative Index eines Elements: Index des Elements minus Startindex des Arrays
  - Beispiel:

```
w : ARRAY [5...10] OF INTEGER;
```

Der relative Index r des dritten Elements von `w`, also `w[7]` ist $r = 7 - 5 = 2$