Unterabschnitt 3.1.13

Module

Begriffsklärung: (Modulsystem)

Ein ***Programmmodul*** fasst mehrere Deklarationen zusammen und stellt sie unter einem Namen zur Verfügung.

Ein Programmmodul sollte Programmierern eine *Modul-Schnittstelle* bereitstellen, die unabhängig von der Implementierung dokumentiert und benutzbar ist.

Module in Haskell

Vereinfacht dargestellt, hat ein Haskell-Modul die Form:

```
module <Modulname> ( <kommagetrennte Liste
                      exportierter Programmelemente>
                   ) where
import <Modul1>
...
import <Moduln>
<Deklarationen des Moduls>
```

Dabei gilt:

- Die Liste der importierten und exportierten Programmelemente kann leer sein.
- Fehlt die Exportliste einschließlich der Klammern, werden alle in dem Modul deklarierten Programmelemente exportiert.

Begriffsklärung: (Programm, die 2.)

Ein ***Haskell-Programm*** (vgl. F. 239) besteht aus einer Menge von Modulen, wobei ein Modul

- den Namen `Main` haben muss und
- in diesem Modul der Name `main` deklariert sein muss.

Die Programmausführung beginnt mit der Ausführung der Deklaration von `main`.

Beispiel: (Haskell-Module)

```
module Alster where

avalue = 7;

data Art = Painting | Design | Sculpture

ache Design = False
ache _      = True
```


Beispiel: (Haskell-Module) (2)

```
module Breg where

import Alster

data Broterwerb = Designer | Maler | Bildhauer

beruf :: Art -> Broterwerb
beruf Design    = Designer
beruf Painting  = Maler
beruf Sculpture = Bildhauer

bflag = (ache Painting)
```


Beispiel: (Haskell-Module) (3)

```
module Main where

import Breg

main = print bflag
```

Beachte:
Programmelemente aus `Alster`, z.B. `avalue`, sind nicht sichtbar in Modul `Main`.

Beispiel: (Modul-Schnittstelle)

```
module Environment (Env, emptyEnv, insertI, insertB,
            lookUp, delete, IBValue(Intv, Boolv, None)
           ) where

emptyEnv :: Env
-- leere Bezeichnerumbegung

insertI :: String -> Int -> Env -> Env
-- (insertI bez i e)  traegt die Bindung (bez,i)
-- in die Umgebung e ein

insertB :: String -> Bool -> Env -> Env
-- (insertB bez b e)  traegt die Bindung (bez,b)
-- in die Umgebung e ein
```

Beispiel: (Modul-Schnittstelle) (2)

```
lookUp :: String -> Env -> IBValue
-- (lookUp bez e)  liefert den Wert v der ersten
-- gefundenen Bindung (bez,v) mit Bezeichner bez

delete :: String -> Env -> Env
-- (delete bez e)  loescht alle Bindungen (bez,_)
-- mit Bezeichner bez
```


Beispiel: (Modul-Implementierung)

```
-- Modulimplementierung (Fortsetzung von Environment)

data IBValue = Intv  Int
             | Boolv Bool
             | None
     deriving (Eq, Show)

type Env = [ (String,IBValue) ]

emptyEnv = []

insertI bez i e = (bez,Intv i):e

insertB bez b e = (bez,Boolv b):e
```


Beispiel: (Modul-Implementierung) (2)

```
lookUp bez []    =  None
lookUp bez ((bz,val):e)
   | bez == bz   =  val
   | otherwise   =  lookUp bez e

delete bez []    =  []
delete bez ((bz,val):e)
   | bez == bz   =  delete bez e
   | otherwise   =  (bz,val):(delete bez e)
```


Unterabschnitt 3.1.14

Zusammenfassung von 3.1

Zusammenfassung von 3.1

Begriffe und Sprachmittel wie Ausdruck, Bezeichner, Vereinbarung, Wert, Typ, Muster, Modul, ...

Wichtige Programmier- und Modellierungskonzepte:

- Basisdatenstrukturen
- rekursive Funktionen
- rekursive Datentypen (insbesondere Listen)
- Ein- und Ausgabe