Heutiges Thema...

Mehr über Haskell, insbesondere über...

- Funktionen
  - ...und darüber wie man sie definieren/notieren kann
    - ⇝ Notationelle Varianten (siehe Vorlesungsteil 1)
    - ⇝ Klammereinsparungsregeln in Funktionssignaturen
    - ⇝ Abseitsregel und Layout-Konventionen
  - Klassifikation von Rekursionstypen
  - Anmerkungen zu Effektivität und Effizienz
  - Komplexitätsklassen

Hinweis: ...die Punkte ab Abseitsregel und Layout-Konventionen werden erst in der kommenden Vorlesung am Do, 03.11.2005 besprochen!

Klammereinsparungsregeln in Funktionssignaturen

Konvention (von essentieller Bedeutung):

- Der *Typkonstruktor* → ist *rechtsassoziativ*!

Das bedeutet:

- Die Funktionssignatur

  ```
  f :: Int -> Float -> Int -> String -> Char
  ```

  steht abkürzend für

  ```
  f :: (Int -> (Float -> (Int -> (String -> Char))))
  ```

- Wann immer eine abweichende Klammerung intendiert ist, *muss* explizit geklammert werden!
  (*vgl. Klammereinsparungsregeln bei arithmetischen Ausdrücken*)

Funktionen und ihre Signaturen (1)

Zur Veranschaulichung, noch konkreter:

Die Signaturen der Funktionen f

```
f :: Int -> (Int -> Int)
```

(aufgrund der Klammereinsparungsregeln gleichbedeutend mit der ungeklammerten Kurzform `f :: Int -> Int -> Int`) und g

```
g :: (Int -> Int) -> Int
```

sind *grundsätzlich verschieden* und *unbedingt auseinanderzuhalten*!

Funktionen und ihre Signaturen (2)

Warum?

- f ist eine Funktion, die ganze Zahlen auf Abbildungen ganzer Zahlen in sich abbildet.
- g ist eine Funktion, die Abbildungen ganzer Zahlen in sich auf ganze Zahlen abbildet.

Zur Übung:

- Überlegen Sie sich, ob die Funktion + (Addition auf ganzen Zahlen) dem Signaturschema von f oder dem von g folgt.

Funktionen und ihre Signaturen (3)

Ein weiteres Beispiel, noch konkreter und noch ein wenig komplexer...

Mit folgenden Deklarationen für f und g

```
f :: Int -> (Int -> Int -> Int)
f 1 = (+)
f 2 = (-)
f 3 = (*)
f _ = div

g :: (Int -> Int -> Int) -> Int
g h = h 6 3
```

...sind folgende Aufrufe mit den angegebenen Resultaten möglich:

```
Main> f 1 2 3                        kurz fuer: (((f 1) 2) 3)
5

Main> f 3 2 3
6

Main> g (+)
9

Main> g (*)
18
```

Funktionen und ihre Signaturen (4)

Beachte: ...in g sind die Argumente 6 und 3 fest vorgegeben. Betrachte deshalb jetzt die folgende “Erweiterung” k von g, die das vermeidet:

```
k :: (Int -> Int -> Int) -> Int -> Int -> Int
k h x y = h x y
```

...was aufgrund der Klammereinsparungsregeln gemäß der Rechtsassoziativität von → abkürzend steht für:

```
k :: ((Int -> (Int -> Int)) -> (Int -> (Int -> Int)))
k h x y = h x y
```

Für k sind jetzt folgende Aufrufe mit variablen Argumenten möglich:

```
Main> k (*) 3 5
15

Main> k (+) 17 4
21
```


Funktionen und ihre Signaturen (5)

Zur Übung:
Vergleichen Sie die Deklaration der Funktion f

```
f :: Int -> (Int -> Int -> Int)
f 1 = (+)
f 2 = (-)
f 3 = (*)
f _ = div
```

...mit der Deklaration ihrer naheliegenden “Umkehrung” g:

```
g :: (Int -> Int -> Int) -> Int
g (+) = 1
g (-) = 2
g (*) = 3
g div = 42
g _   = 99
```

- Was beobachten Sie, wenn Sie die Funktionen f und g aufrufen?
- Haben Sie (schon) eine Erklärung dafür?

Funktionen und ihre Signaturen (6)

Bleiben Sie auch an folgender Frage dran...

- Warum möglicherweise sind die Klammereinsparungsregeln für →

  ```
  f :: Int -> Int -> Int -> Int
  ```

  zugunsten der *Rechtsassoziativität* von →

  ```
  f :: (Int -> (Int -> (Int -> Int)))
  ```

  und nicht der *Linksassoziativität* gefallen?

  ```
  f :: (((Int -> Int) -> Int) -> Int)
  ```

Funktionen und ihre Signaturen (7)

In jedem Falle gilt:

Die Einsicht in den Unterschied von

```
f :: Int -> Int -> Int -> Int
```

...aufgrund der *Rechtsassoziativität* von → abkürzend für die vollständig, aber nicht überflüssig geklammerte Version

```
f :: (Int -> (Int -> (Int -> Int)))
```

und von

```
f :: (((Int -> Int) -> Int) -> Int)
```

ist *zentral* und von *essentieller* Bedeutung!

Ohne diese Einsicht ist erfolgreiche Programmierung im funktionalen Paradigma

- nicht möglich
- oder allenfalls Zufall!

Zum ersten Aufgabenblatt...

- ...erhältlich seit gestern im Web unter folgender URL
  `http://www.complang.tuwien.ac.at/knoop/fp185161_ws0506.html`
- Abgabe: Mo, den 31.10.2005, 12:00 Uhr
- "Nachträgliche" Abgabe: Mo, den 07.11.2005, 12:00 Uhr

Vorschau:
Ausgabe des...

- zweiten Aufgabenblatts: Fr, den 28.10.2005
  ...Abgabetermine: Fr, 04.11.2005, und Fr, 11.11.2005
- dritten Aufgabenblatts: Fr, den 04.11.2005
  ...Abgabetermine: Mo, 14.11.2005, und Mo, 21.11.2005

Vorschau auf die nächsten Vorlesungstermine...

- *Do, 27.10.2005: Keine Vorlesung!*
- Do, 03.11.2005, Vorlesung von 16:30 Uhr bis 18:00 Uhr im Radinger-Hörsaal
- *Do, 10.11.2005: Keine Vorlesung!*
- Do, 17.11.2005, Vorlesung von 16:30 Uhr bis 18:00 Uhr im Radinger-Hörsaal
- Do, 24.11.2005, Vorlesung von 16:30 Uhr bis 18:00 Uhr im Radinger-Hörsaal