Heutiges Thema

- Elementare Datentypen und Listen
- Ausdrücke, Ausdrucksauswertung und Auswertungsstrategien

(Ausgew.) Elementare Datentypen (1)

Wahrheitswerte

Typ	`Bool`	Wahrheitswerte
Konstanten	`True :: Bool`	Symbol für "wahr"
	`False :: Bool`	Symbol für "falsch"
Operatoren	`&& :: Bool -> Bool -> Bool`	logisches und
	`|| :: Bool -> Bool -> Bool`	logisches oder
	`not :: Bool -> Bool`	logische Negation

Elementare Datentypen (2)

Ganze Zahlen

Typ	`Int`	Ganze Zahlen(endl. Ausschn.)
Konstanten	`0 :: Int`	Symbol für "0"
	`-42 :: Int`	Symbol für "-42"
	`2147483647 :: Int`	Symbol für "maxInt"
	...	
Operatoren	`+ :: Int -> Int -> Int`	Addition
	`* :: Int -> Int -> Int`	Multiplikation
	`^ :: Int -> Int -> Int`	Exponentiation
	`- :: Int -> Int -> Int`	Subtraktion (Infix)
	`- :: Int -> Int`	Vorzeichenwechsel (Prefix)
	`div :: Int -> Int -> Int`	Division
	`mod :: Int -> Int -> Int`	Divisionsrest
	`abs :: Int -> Int`	Absolutbetrag
	`negate :: Int -> Int`	Vorzeichenwechsel

Elementare Datentypen (3)

Ganze Zahlen (fortgesetzt)

Relatoren	`> :: Int -> Int -> Bool`	echt größer
	`>= :: Int -> Int -> Bool`	größer gleich
	`== :: Int -> Int -> Bool`	gleich
	`/= :: Int -> Int -> Bool`	ungleich
	`<= :: Int -> Int -> Bool`	keiner gleich
	`< :: Int -> Int -> Bool`	echt kleiner

...die Relatoren `==` und `/=` sind auf Argumente beliebiger Typen anwendbar, beispielsweise auch auf Wahrheitswerte (Stichwort: Overloading!)

Elementare Datentypen (4)

Zeichen

Typ	`Char`	Zeichen (Literal)
Konstanten	`'a' :: Char`	Symbol für "a"
	...	
	`'Z' :: Char`	Symbol für "Z"
	`'\t' :: Char`	Tabulator
	`'\n' :: Char`	Neue Zeile
	`'\\' :: Char`	Symbol für "backslash"
	`'\'' :: Char`	Hochkomma
	`'\"' :: Char`	Anführungszeichen
Operatoren	`ord :: Char -> Int`	Konversionsfunktion
	`chr :: Int -> Char`	Konversionsfunktion

Elementare Datentypen (5)

Gleitkommazahlen

Typ	`Float`	Gleitkommazahlen (endlicher Ausschnitt)
Konstanten	`0.123 :: Float`	Symbol für "0,123"
	`-42.4711 :: Float`	Symbol für "-42,4711"
	`123.6e-2 :: Float`	$123{,}6 \times 10^{-2}$
	...	
Operatoren	`+ :: Float -> Float -> Float`	Addition
	`* :: Float -> Float -> Float`	Multiplikation
	...	
	`sqrt :: Float -> Float`	(pos.) Quadratwurzel
	`sin :: Float -> Float`	sinus
	...	
Relatoren	`== :: Float -> Float -> Bool`	gleich
	`/= :: Float -> Float -> Bool`	ungleich
	...	

Elementare Datentypen und mehr...

Zeichenreihen

Typ	`String`	Zeichenreihen
	`type String = [Char]`	Deklaration (als Liste von Zeichen)
Konstanten	`"Haskell" :: String`	Zeichenr. für "Haskell"
	`""`	Leere Zeichenreihe
	...	
Operatoren	`++ :: String -> String -> String`	Konkatenation
Relatoren	`== :: String -> String -> Bool`	gleich
	`/= :: String -> String -> Bool`	ungleich

Listen

Intuition

Listen ...fassen eine beliebige/unbestimmte Zahl von Werten gleichen Typs zusammen.

Beispiele:

- Listen ganzer Zahlen
  `[2,5,12,42] :: [ Int ]`
- Listen von Wahrheitswerten
  `[True,False,True] :: [ Bool ]`
- ...

Listen

Bereits vordefiniert...

```
Type String = [Char]
```

Auch möglich...

- Listen von Listen
  `[[2,4,23,2,5],[3,4],[],[56,7,6,]] :: [ [Int] ]`
- Listen von Paaren
  `[(3.14,42.0),(56.1,51.3)] :: [ (Float,Float)]`
- ...
- *Ausblick*: Listen von Funktionen
  `[fact, abs, negate] :: [ Integer -> Integer ]`

Mehr über Listen 1

length :: [a] −> Integer
length [] = 0
length (x:xs) = 1 + length xs

length [1, 2, 3] ⇒ 3 ([1,2,3] = 1:2:3:[])
length ['a','b','c'] ⇒ 3
length [[1],[2],[3]] ⇒ 3

head :: [a] −> a tail :: [a] −> [a]
head (x:xs) = x tail (x:xs) = xs

['h','e','l','l','o'] = "hello"
"hello" ++ " world" ⇒ "hello world"

Mehr über Listen 2

- Spezialfälle (i.w. für Listen über Zahlen und Zeichen)
  ...`[2 .. 6]` kurz für `[2,3,4,5,6]`
  ...`[11,9 .. 2]` kurz für `[11,9,7,5,3]`
  ...`['a','d' .. 'j']` kurz für `['a','d','g','j']`
  ...`[0.0,0.3 .. 1.0]` kurz für `[0.0,0.3,0.6,0.9]`
- Listenkomprehension
  ...ein erstes Beispiel:
  `[3*n | n <- list]` kurz für `[3,6,9,12]`, falls `list` vom Wert `[1,2,3,4]` ist.

  ⇝ Listenkomprehension ist ein sehr ausdruckskräftiges Sprachkonstrukt!

Ausdrücke mit Typen

5 :: Int	−− ganze Zahl (32-bit)
::	(Haskell-Report: $-2^{29}..2^{29}-1$)
5 :: Integer	−− ganze Zahl (beliebig)
'a' :: Char	−− Zeichen
[1,2,3] :: [Integer]	−− Liste ganzer Zahlen
('b',4) :: (Char, Integer)	−− Paar bzw. Tupel
\x−>x+1 :: Integer −> Integer	−− Funktion
\x y−>(x,y) :: a −> b −> (a,b)	−− generische Funktion
\x−>x+1 :: Num a=>a−>a	−− Typ-/Klasseneinschränkung

Funktionsdefinition mit Typdeklaration

...in `.hs` Datei:

e :: Float
e = 2.71828

simple :: Integer−>Integer−>Integer−>Integer
simple x y z = x ∗ (y + z)

fact :: Integer −> Integer
fact n = if n == 0 then 1 else n ∗ fact (n − 1)

Auf Einrückungen achten! Stichwort: Abseitsregel!

Auswerten von Ausdrücken und Funktionen...

Fundamental...

- Expandieren
- Simplifizieren

Einfache Ausdrücke und Funktionen (1)

3 ∗ (9 + 5) ⇒ 3 ∗ 14 ⇒ 42 oder
3 ∗ (9 + 5) ⇒ 3 ∗ 9 + 3 ∗ 5 ⇒ 27 + 3 ∗ 5 ⇒ 27 + 15 ⇒ 42

simple x y z = x ∗ (y + z)
simple 3 9 5 ⇒ 3 ∗ (9 + 5) ⇒ 3 ∗ 14 ⇒ 42

Einfache Ausdrücke und Funktionen (2)

fact n = if n == 0 then 1 else (n ∗ fact (n − 1))

fact 2 ⇒ if 2 == 0 then 1 else (2 ∗ fact (2 − 1))
⇒ 2 ∗ fact (2 − 1)
⇒ 2 ∗ fact 1
⇒ 2 ∗ (if 1 == 0 then 1 else (1 ∗ fact (1 − 1)))
⇒ 2 ∗ (1 ∗ fact (1 − 1))
⇒ 2 ∗ (1 ∗ fact 0)
⇒ 2 ∗ (1 ∗ (if 0 == 0 then 1 else (0 ∗ fact (0 − 1))))
⇒ 2 ∗ (1 ∗ 1)
⇒ 2 ∗ 1
⇒ 2

Zur Syntax von Haskell... hier Namen

Name ist — Buchstabe gefolgt von Buchstaben und Ziffern
— Infix-Operator als Folge von Sonderzeichen

Funktionsnamen, Variablen beginnen mit Kleinbuchstaben — Großbuchstaben für Typen, Konstruktoren, Module reserviert

Leerzeichen und Klammern zur Trennung von Namen

Funktionsargumente durch Leerzeichen getrennt

Leerzeichen binden stärker als Operatoren

Trennung: — Funktionsdef., Typdeklarationen in Dateien
— Auswertung von Ausdrücken im Interpreter

Ein- und mehrstellige Funktionen

Funktionen in Haskell sind grundsätzlich einstellig. Mehrstelligkeit ist implizit realisiert. Hier am Beispiel *anonymer* Funktionen:

```
\x y z->(x * (y + z)) 3 9 5  ⇒ \y z->(3 * (y + z)) 9 5
                             ⇒ \z->(3 * (9 + z)) 5
                             ⇒ 3 * (9 + 5)
                             ⇒ 3 * 14
                             ⇒ 42
```

Schrittweise Berechnung direkt darstellbar:

```
      simple = \x y z->(x * (y + z))
    simple x = \y z->(x * (y + z))
  simple x y = \z->(x * (y + z))
simple x y z = x * (y + z)
```

Stichwort: Currying (...später mehr dazu)

Nächste Vorlesungstermine...

- **Heute**: Di, 19.10.2004, 18-19 Uhr, HS 8 Heinz Parkus, Hauptgebäude, Karlsplatz 13

Vorschau auf weitere Vorlesungstermine...

- Di, 26.10.2004 und Di, 02.11.2004 keine Vorlesung
- Ab Di, 09.11.2004, jeden Dienstag von 13-14 Uhr im Treitl-Hörsaal