

Vorsemesterkurs Informatik

Sommersemester 2011

Grundlagen der Programmierung in Haskell

SoSe 2011

Übersicht

- 1 Ausdrücke und Typen
- 2 Funktionen
- 3 Rekursion
- 4 Listen
- 5 Paare und Tupel

Programmieren in Haskell

Haskell-Programmieren:

- Im Wesentlichen formt man **Ausdrücke**
- z.B. arithmetische Ausdrücke $17*2+5*3$
- Ausführung: Berechnet den Wert eines Ausdrucks
- Ausdrücke **zusammensetzen** durch:
 - **Anwendung** von Funktionen auf Argumente,
dabei sind **Werte** die kleinsten „Bauteile“

Programmieren in Haskell (2)

- In Haskell hat jeder Ausdruck (und Unterausdruck) einen **Typ**
- Typ = Art des Ausdrucks
z.B. Buchstabe, Zahl, Liste von Zahlen, Funktion, ...
- Die Typen müssen zueinander passen:
Z.B. **verboten**

`1 + "Hallo"`

Die Typen passen nicht zusammen (Zahl und Zeichenkette)




Typen

Im GHCi Typen anzeigen lassen:

```
Prelude> :type 'C'   
'C' :: Char
```

Sprechweise: „'C' hat den Typ Char“



- Char ist der Typ in Haskell für Zeichen (engl. Character)
- Typnamen beginnen immer mit einem Großbuchstaben
- Im GHCi: `:set +t` führt dazu, dass mit jedem Ergebnis auch dessen Typ gedruckt wird.

```
*Main> :set +t   
*Main> zwei_mal_Zwei   
4  
it :: Integer  
*Main> oft_fuenf_addieren   
55  
it :: Integer
```

Typen (2)

- Der Typ `Integer` stellt beliebig große ganze Zahlen dar
- Man kann Typen auch selbst angeben:




Schreibweise `Ausdruck :: Typ`




```
*Main> 'C' :: Char 
'C'
it :: Char
*Main> 'C' :: Integer 

<interactive>:1:0:
  Couldn't match expected type 'Integer'
  against inferred type 'Char'
  In the expression: 'C' :: Integer
  In the definition of 'it': it = 'C' :: Integer
```

Wahrheitswerte: Der Datentyp Bool

- Werte (Datenkonstrukturen) vom Typ `Bool`:
 - `True` steht für „wahr“
 - `False` steht für „falsch“
- Operationen (Funktionen):
 - Logische Negation: `not`: liefert `True` für `False` und `False` für `True`
 - Logisches Und: `a && b`: nur `True`, wenn `a` und `b` zu `True` auswerten
 - Logisches Oder: `a || b`: `True`, sobald `a` oder `b` zu `True` auswertet.

```
*Main> not True   
False  
it :: Bool  
*Main> not False   
True  
it :: Bool  
*Main> True && True   
True  
it :: Bool
```

```
*Main> False && True   
False  
it :: Bool  
*Main> False || False   
False  
it :: Bool  
*Main> True || False   
True  
it :: Bool
```

Ganze Zahlen: Int und Integer

- Der Typ **Int** umfasst ganze Zahlen **beschränkter** Größe (je nach Rechner, z.B. -2^{31} bis $2^{31} - 1$)
- Der Typ **Integer** umfasst ganze Zahlen **beliebiger** Größe

Ganze Zahlen: Int und Integer

- Der Typ **Int** umfasst ganze Zahlen **beschränkter** Größe (je nach Rechner, z.B. -2^{31} bis $2^{31} - 1$)
- Der Typ **Integer** umfasst ganze Zahlen **beliebiger** Größe
- Darstellung der Zahlen ist identisch z.B. 1000
- Defaulting: Integer, wenn es nötig ist, sonst offenlassen:

Ganze Zahlen: Int und Integer

- Der Typ `Int` umfasst ganze Zahlen **beschränkter** Größe (je nach Rechner, z.B. -2^{31} bis $2^{31} - 1$)
- Der Typ `Integer` umfasst ganze Zahlen **beliebiger** Größe
- Darstellung der Zahlen ist identisch z.B. 1000
- Defaulting: `Integer`, wenn es nötig ist, sonst offenlassen:

```
Prelude> :type 1000
1000 :: (Num t) => t
Prelude> :set +t
Prelude> 1000
1000
it :: Integer
```

- In etwa 1000 ist vom Typ `t`, wenn `t` ein **numerischer** Typ ist.
- Genauer: `(Num t) =>` ist eine sog. **Typklassenbeschränkung**
- `Int` und `Integer` sind numerische Typen (haben Instanzen für `Num`)