Informatik I - Programmierung
Globalübung
13.01.2003

Hugs98
Currying

Thomas Weiler

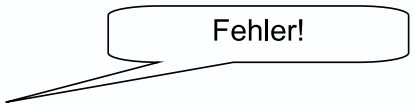
Fachgruppe Informatik
RWTH Aachen

Inhalt

- **Hugs98**
  - Bedienung
  - Typen
  - Fehlermeldungen
- **Currying**

Verwendung des Hugs98 Interpreters

- Der Interpreter erwartet nach dem Start als Eingabe einen auszuwertenden Ausdruck oder ein Kommando
- Prelude> 42
  42
- Prelude> 4 * 11 – 2
  42
- Prelude> sqrt 2
  1.41421
- Prelude> max 3 -5
  ERROR



- Prelude> max 3 (-5)
  3

Kommandos - 1

- Wichtig: Jedes Kommando startet mit einem Doppelpunkt!

:load <filenames>	Lädt ein Modul aus der angegebenen Datei
:load	Entlädt alle Dateien außer *prelude*
:also <filenames>	Liest weitere Module
:reload	Wiederholt das letzte *load* Kommando
:edit <filename>	Öffnet die angegebene Datei im Editor
:edit	Öffnet das zuletzt geladene Modul im Editor
:type <expr>	Gibt den Typ des Ausdrucks aus

Kommandos – 2

:?	Gibt eine Liste der Kommandos aus
:set <options>	Setzt Kommandozeilenoptionen
:set	Gibt eine Hilfe zu den Kommandozeilenoptionen aus
:find <name>	Öffnet das Modul im Editor, welches die Definition von <name> enthält
:!command	Führt command in einer BS-Shell aus
:cd dir	Wechselt das aktuelle Verzeichnis
:gc	Erzwingt eine garbage collection
:version	Gibt die Version des Interpreters aus
:quit	Beendet den Interpreter

Kommandozeilenoptionen

- **Eingabeformat** :set [+|-]<schalter>

  Beispiel:

```
Prelude>3+4
7
Prelude>:s +t
Prelude>3+4
7 :: Integer
```

- t Gib Typ nach der Berechnung aus
- f Beende die Auswertung bei Auftreten eines Fehlers
- k Gibt etwas verständlichere/längere Fehlermeldungen aus
- Umgebungsvariablen:
  - pstr Setzt den Eingabeprompt auf str
  - Pstr Setzt den Suchpfad für Module auf str
  - Estr Setzt den Editor auf str
- **Beispiel:**

  :set +Enotepad

+ oder – ist egal, muss aber angegeben werden

Haskell-Programm

- Sammlung von Definitionen
- Bezeichner von Funktionen und Variablen müssen mit einem Kleinbuchstaben beginnen
- Einzeiliger Kommentar:

```
-- Dies ist ein einzeiliger Kommentar
```

- Mehrzeiliger Kommentar:

```
{-
        Dies ist ein mehrzeiliger Kommentar

-}
```


Haskell und Typen

- Haskell ist stark getypt, d.h. zur Laufzeit können keine Fehler durch Anwendung von Funktionen auf Argumente des falschen Typs entstehen
- Die Typen aller Ausdrücke sind zur Übersetzungszeit bekannt (statisches Typsystem).
- Basistypen, z.B. Bool, Int, Char
- Funktionstypen, z.B. even :: Int -> Bool
- Der Typ einer selbstdefinierten Funktion kann jedoch auch in einer Typdeklaration angegeben werden
  - Interpreter überprüft, ob die Funktion wirklich den deklarierten Typ hat
  - Erhöht die Verständlichkeit für den Leser
  - Bei Typfehlern sind die Fehlermeldungen aussagekräftiger

Vordefinierte Typen - 1

Wahrheitswerte

- **Typ**: Bool
- **Werte**: False, True
- **Funktionen**:
  (&&) :: Bool -> Bool -> Bool
  (||) :: Bool -> Bool -> Bool
  not :: Bool -> Bool
  } **Infixoperatoren**
- **Vergleichsoperatoren** existieren für alle vordefinierten Typen (außer Funktionen): <, <=, == (gleich), >=, >, /= (ungleich)

Zahlen

- **Typen:** Int, Integer, Float, Double, Rational, Complex Float, Complex Double,…
- **Einige Funktionen:**
  Zahlen: +, - , *, negate, abs, signum
  Int, Integer: div, mod, even, odd
  Real, Double: exp, log, sqrt, sin

Int*:* beschränkte Genauigkeit (32 bit)
Integer*:* unbeschränkte Genauigkeit
Float*:* einfache Genauigkeit
Double*:* doppelte Genauigkeit

Vordefinierte Typen - 2

Zeichen

- **Typ:** Char
- **Werte in einfachen Anführungszeichen. Beispiele:**
  ‘a‘, ‘4‘, ‘\n‘ (newline)
- **Einige Funktionen:**
  ord :: Char -> Int
  chr :: Int -> Char
  isDigit, isUpper :: Char -> Bool
  toUpper, toLower :: Char -> Char

Zeichenketten

- **Typ**: String (identisch mit [Char])
- **Werte**: z.B. “Hallo“
- **Einige Funktionen:**
  reverse :: String -> String
  error :: String -> a
- **error “Fehlermeldung“** hat beliebigen Typ. Führt zum Abbruch der Berechnung und Ausgabe der Fehlermeldung

Listen

- Listen sind die zentralen Datenstrukturen in funktionalen Programmen.
- Eine Liste ist eine Folge von Werten des gleichen Typs.
  - Tupel: Typen der Elemente können verschieden sein.

    Beispiel: (1, ‘a‘, True) :: (Int, Char, Bool)
- Funktionen mit Listen:

(++) :: [a] -> [a] -> [a]	verbindet zwei Listen
head, last :: [a] -> a	erstes/letztes Element der Liste
tail, init :: [a] -> [a]	Rest/vorderer Teil der Liste
length :: [a] -> Int	Listenlänge
(!!) :: [a] -> Int -> a	n-tes Elements; Start bei 0

Fehlermeldungen

- Fehlermeldungen in Hugs helfen nicht immer unbedingt bei der Fehlersuche (http://www.cs.kent.ac.uk/people/staff/sjt/craft2e/errors/allErrors.html)
- Beispiele:
  - > 3 ´div´ 4
    ERROR - Undefined variable "´div´“

    > 3 ‘div‘ 4
    ERROR - Improperly terminated character constant
  - Problem: Wird eine Funktion als Infix-Operator verwendet, so muss der Name in Backquotes (Accent grave) eingeschlossen werden.
  - plus :: a -> a -> a
    plus x y = x + y
    ERROR "Test.hs":22 - Inferred type is not general enough
    *** Expression : plus
    *** Expected type : a -> a -> a
    *** Inferred type : Int -> Int -> Int
  - Problem: Der Typ der Funktion (Int -> Int -> Int) ist eingeschränkter als der deklarierte Typ (a -> a -> a)

Currying - 1

- In der Mathematik werden Argumente einer Funktion meistens zu einem Tupel zusammengefasst

  Beispiel (Potenzierung):

  $$power(basis, exponent) = basis^{exponent}$$

- Eine *curried Function* verbraucht im allgemeinen das erste Argument und liefert wiederum eine Funktion, die das zweite Argument verbraucht usw.

  Beispiel (Potenzierung):

  $$(power(basis))(exponent) = basis^{exponent}$$

Currying - 2

- Currying ist eine Operation, die auf eine Funktion angewendet wird, welche mehr als ein Argument hat.
- Das Ergebnis der Currying Operation angewendet auf eine Funktion f(x,y) ist eine Funktion (g(x))(y), so dass gilt:

  $$f(x, y) = ((g(x)) (y))$$

- Nutzen: Insbesondere bei der Verwendung von Funktionen höherer Ordnung (Funktionen, mit Funktionen als Argumente)

  Beispiel: Zu allen Elementen einer Liste 3 addieren

Haskell Brooks Curry

- Geboren: 12. September 1900, Massachusetts
- Gestorben: 1. September 1982, Pennsylvania
- Studierte in Harvard
- Promovierte in Göttingen bei Hilbert über *Grundlagen der kombinatorischen Logik*
- Lehrte in Harvard, Princeton und Pennsylvania State University
- 1966 Chair of mathematics in Amsterdam