Praktische Informatik 1 WiSe 98/99

Prof. Bernd Krieg-Brückner
Shi Hui
Kolyang
Axel Lankenau
Christoph Lüth
Lutz Zegartowski

3. Arbeitsblatt

Ausgabe: 19.11.1998
Bearbeitungszeit: drei Tutorien

Lektüre

Die Schwerpunkte dieser Lektüre sind Listen, rekursive Datentypen, Pattern–Matching und abstrakte Datentypen (ADT). Listen werden in den Abschnitten 4.1–4.3 und 4.6 erklärt. Zum Thema Pattern–Matching lest bitte Abschnitt 4.4 im Haskell–Buch. Informationen über rekursive Datentypen findet Ihr in Kapitel 10.2. Abstrakte Datentypen werden in Kapitel 12 des Buches behandelt.

Leitfragen

1. Listen...
   (a) Wie unterscheiden sich Listen von Arrays, die Du aus anderen Programmiersprachen kennst? Was sind die Vor– und was die Nachteile?
   (b) Wie stellt man Listen in Haskell dar?
   (c) Gibt es unendliche Listen? Was heißt `[0..9]` oder `[1,3..100]`?
   (d) Was versteht man unter *list comprehension*?
   (e) Welche Funktionen auf Listen sind Dir bekannt?
2. Pattern–Matching...
   (a) Welche Ausdrücke darf man als Pattern verwenden? Was heißt Pattern–Matching?
   (b) Wie definiert man eine Funktion durch Pattern–Matching? Was ist der Unterschied zwischen dieser Art der Funktionsdefinition und der durch bedingte Gleichungen? Welche Art der Definition bevorzugst Du?
3. Datentypen...
   (a) Was ist ein Aufzählungstyp? Was ist ein rekursiver Datentyp? Was ist der Unterschied?
   (b) Kannst Du ein reales Objekt mit derartigen Datentypen modellieren?
   (c) Was ist ein abstrakter Datentyp? Warum sind solche Datentypen abstrakt?
   (d) Kannst Du ein bekanntes Objekt mit einem abstrakten Datentyp modellieren?

Anwendungsaufgaben

1. **Einfache Funktionen auf Listen**
   Überlege Dir einfache Funktionen auf Listen und implementiere sie in Haskell. Beispiel: Berechnung der Länge einer Liste mit der Funktion `laenge :: [a] -> Int`.
2. **Sieb des Eratosthenes**
   Schreibe eine Haskell–Funktion, die alle Primzahlen kleiner n berechnet. Verwende den als *Sieb des Eratosthenes* bekannten Algorithmus, der sukzessive alle Vielfachen der bereits als Primzahlen bekannten Zahlen aus der Liste `[2..n]` streicht. Übrig bleibt dann die gewünschte Liste der Primzahlen.

3. ***List comprehension***
   Definiere die Liste aller Tripel (a, b, c) mit $a, b, c \in \{1, ..., 100\}$, die die pythagoräische Gleichung $a^2 + b^2 = c^2$ erfüllen. Überlege Dir weitere Anwendungen der *list comprehension*!

4. **Verarbeitung von Zeichenfolgen**
   Zeichenfolgen haben offensichtlich eine zentrale Bedeutung für Textverarbeitungssysteme, die eine Vielzahl von Möglichkeiten für die Bearbeitung von Text bereitstellen. Eine grundlegende Operation mit Zeichenfolgen ist die Textsuche: Gegeben seien eine Text–Zeichenfolge und eine Muster–Zeichenfolge; zu suchen ist ein Auftreten des Musters innerhalb des Textes. Implementiere eine Funktion `substring :: String -> String -> Int`.

5. **Übung 12.1 zu Abschnitt 12.1**
   Geben Sie eine Implementierung von `Store` mit Listen, deren Einträge nach den Variablennamen sortiert sind. Warum könnte diese Implementierung besser als die ursprüngliche Implementierung mit unsortierten Listen sein? Gibt es auch Nachteile und wenn ja, welche?

6. **Übung 12.8, 12.10 zu Abschnitt 12.3**
   Erkläre das Verhalten der drei Queue–Modelle (Hinten anfügen, vorne wegnehmen; Vorn anfügen, hinten wegnehmen; Queue–Modellierung durch zwei Listen) bei Durchführung folgender Operationen in der gegebenen Reihenfolge: `add 2`, `add 1`, `remove item`, `add 3`, `remove item`, `add 1`, `add 4`, `remove item`, `remove item`.

   Eine eindeutige Queue darf nur ein einziges Vorkommen jedes Eintrags enthalten. Gib eine Signatur für den ADT von diesen Queues und eine Implementierung der Funktionen `addQ` und `remQ`!