Praktische Informatik 3 WS 2003/04

Berthold Hoffmann <hof>
Christian Maeder <maeder>
Klaus Lüttich <luettich>
Pascal Schmidt <pharao90>
Felix Beckwermert <foetus>
Dennis Walter <dw>
Franck Ngueuleu <nngf>

3. Übungsblatt

Ausgabe: 17.11.2002

Bearbeitungszeit: Zwei Wochen

---

6 *Text Formatierung* *8 Punkte*

Uns liegt ein zu formatierender Text zeilenweise vom Typ `[String]` vor.

1. Der gesamte Text soll unabhängig von der Zeilenstruktur zunächst in eine Liste von Wörtern umgewandelt werden. Dazu kann die vordefinierte Funktion '`words :: String -> [String]`' zu Hilfe genommen werden. (Wörter enthalten keine Leerzeichen.)

2. Ein Wort, das mit einem Punkt (`'.'`) endet, markiert bei uns ein Satzende. Formen Sie eine Ausgangsliste von Zeilen in eine Liste von Sätzen um!

3. Die Wörter eines Satzes sollen nun wieder auf eine oder mehrere (nicht leere) Zeilen verteilt werden, die — maximal gefüllt — eine gewisse Zeilenbreite nicht überschreiten. (Wörter, die länger als die Zeilenbreite sind, bilden eine Zeile für sich.)

4. Die Wörter einer Zeile sollen nun unterschiedlich formatiert ausgegeben werden können: linksbündig, rechtbündig oder zentriert. Implementieren Sie entsprechende Funktionen:

   ```
   type LinePrinter = [String] -> String
   left :: LinePrinter
   right, center :: Int -> LinePrinter
   ```

   Der Parameter vom Typ `Int` ist die Zeilenbreite, die benötigt wird, um den Spielraum für Leerzeichen zu berechnen.

5. Implementieren Sie nun eine Funktion `prettyPrint`, die einen Text in seine Sätze zerlegt und alle Zeilen der Sätze (gemäß Teilaufgabe 3) mit einem Prettyprinter (aus Teilaufgabe 4) formatiert. Das Ergebnis ist wieder vom Typ `[String]`, wobei zwei Sätze jeweils durch eine Leerzeile getrennt werden sollen.

   ```
   prettyPrint :: LinePrinter -> Int -> [String] -> [String]
   ```

Hinweis: Mit Hilfe der Funktion '`unlines :: [String] -> String`' können mehrere Textzeilen zu einem Gesamttext (der dann '`\n`' Escape-Sequenzen enthält) zusammengeführt werden. Durch '`putStrLn`' — nur interaktiv am Kommandoprompt von hugs oder ghci — wird ein Text dann tatsächlich (gemäß seiner Escape-Sequenzen) angezeigt.

7 *Tabellen* — *6 Punkte*

Im folgenden modellieren wir Tabellen als (totale) Abbildungen von Indexpaaren — bestehend aus Zeilen- und Spaltennummer — auf Zelleninhalte vom Typ `String`, die im Zweifelsfall leer sind:

```
type IntPair = (Int, Int)
type Table = IntPair -> String
```

1. Schreiben Sie Funktionen, die eine leere Tabelle erstellt, den Inhalt einer bestimmten Zelle setzt und den Inhalt einer Zelle liefert.

   ```
   empty    :: Table
   set      :: IntPair -> String -> Table -> Table
   get      :: Table -> IntPair -> String
   ```

2. Implementieren Sie zwei Funktionen: die eine extrahiert aus einer Tabelle n Zellen rechts von einem Indexpaar und die andere formatiert jede Zelle rechtsbündig auf eine feste Zellenbreite (1. Argument) und gibt alle Zelleninhalte getrennt durch einen senkrechten Strich `'|'` als eine Zeile aus. (Zu lange Zelleninhalte werden am Ende abgeschnitten.)

   ```
   getRow :: Table -> Int -> IntPair -> [String]
   printRow :: Int -> LinePrinter
   ```

3. Implementieren Sie eine Funktion, die alle Zellen in einem Indexbereich, der durch zwei Indexpaare gegeben ist, als formatierte Zeilen ausgibt.

   ```
   printTable :: Table -> IntPair -> IntPair -> LinePrinter -> [String]
   ```

4. Implementieren Sie eine Funktion, die aus einer Liste von Wortenlisten (`[[String]]`) eine Tabelle ab dem Indexpaar (1,1) erzeugt.

8 *Algebraische Datentypen (ab dem 24.11.03)* — *6 Punkte*

1. Geben Sie einen Aufzählungstyp für die 4 Grundrechenarten an.

2. Geben Sie unter Verwendung des obigen Aufzählungstyps einen rekursiven Datentyp '`Expr a`' für arithmetische Ausdrücke an, deren Werte (vom Typ '`a`' aus der Klasse '`Fractional a`') entweder explizit angegeben werden können oder durch ein Indexpaar auf einen Tabelleneintrag verweisen.

3. Schreiben Sie eine Funktion, die den Wert eines Ausdrucks berechnet. (Der String einer Zelle kann mit der vordefinierten Funktion '`read :: (Read a) => String -> a`' in eine '`Fractional`' Zahl umgewandelt werden.)

   ```
   eval :: (Fractional a, Read a) => Table -> Expr a -> a
   ```

Dieses ist Version 2.0 vom 17. November 2003.