Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion
CAU Kiel, Informatik und Prakt. Mathematik
Dr. Frank Huch

24118 Kiel
Christan-Albrechts-Platz 4
Tel.: 0431 / 880-7277

8. Übung „Funktionale Programmierung“

Abgabe am 8. Juni 2005 vor der Vorlesung

Wichtig: Denken Sie bei allen definierten Funktionen/Typen an die auf dem 1. Übungsblatt formulierten Regeln (z.B. Typsignaturen, Beispielanwendungen und Beispielwerte).

Aufgabe 1 10 Punkte

In dieser Aufgabe sollen Sie eigene Parserkombinatoren mit Fehlermeldungen und Positionsinformationen entwickeln. Verwenden Sie folgende Typen für die Implementierung Ihrer Parserkombinatoren:

```
type Parser a = String -> Pos -> Result a

data Result a = Result a String Pos
              | Error String
  deriving Show

data Pos = Pos Int Int
```

Da wir also keine Liste als Ergebnistypen verwenden, handelt es sich um einen deterministischen Parser. Außerdem arbeitet dieser Parser nicht auf beliebigen Tokenfolgen, sondern auf Strings.
Passen Sie die in der Vorlesung vorgestellten (nicht-monadischen) Parserkombinatoren für diesen Parser an. Ihr Parser soll Fehlerwerte der Form:

```
Error "unexpected symbol 'b' in line 1 column 3"
```

erzeugen.
Implementieren Sie einen Parser, der einfache imperative Programme zu folgender Grammatik (hier Backus-Nauer-Form) parst (Startsymbol: *Stms*):

Stms	::=	*Stm* `;` *Stms*	*Stm*	::=	`begin` *Stms* `end`
	|	*Stm*		|	`if` *Exp* `then` *Stm*
Exp	::=	*ident*		|	`while` *Exp* `do` *Stm*
	|	*digit*		|	*ident* `:=` *Exp*

Hierbei repräsentiert das Terminal *ident* alle Identifier und das Terminal *digit* alle Zahlen. Wie in Programmiersprachen üblich sollen beim parsen Whitespaces (Leerzeichen und Zeilenumbrüche) ignoriert werden. Für die Fehlermeldungen sollen diese aber natürlich berücksichtigt werden.

Aufgabe 2 10 Punkte

Vergleicht man die monadischen Parserkombinatoren mit der in der Vorlesung vorgestellten Zustandsmonade, so gibt es große Ähnlichkeiten: die Token können auch als Zustand des Parser interpretiert werden.

a) Erweitern Sie den Monadischen Parser um einen weiteren Zustand, in welchem beliebige Werte während des Parsens gespeichert werden können. Der allgemeine Parser sollte dann den Typen `Parser t s a` haben, wobei `t` der Tokentyp, `s` der Zustandstyp und `a` der Ergebnistyp (der Parsermonade) sind.

   Implementieren Sie einen Parser für die Sprache $\{a, b\}^*$, der die Anzahl der a-Symbole im Zustand und die Anzahl der b-Symbole im Ergebnis zählt.

b) Erweitern Sie die Zustandsmonade aus der Vorlesung um nicht-deterministische Verzweigung. Definieren Sie hierzu einen Operator `<|>`, der eine Berechnung aufspaltet.

   Verwenden Sie die definierte nichtdeterministische Zustandsmonade um die Blätter eines Baumes in allen möglichen Reihenfolgen zu nummerieren, sprich die Zahlen von 1 bis Anzahl der Blätter des Baumes auf die Blätter zu verteilen. Modifizieren Sie für Ihre Implemententierung die in der Vorlesung präsentierte Funktion `number`. Bsp.:

```
Main> number (Node (Leaf ())(Leaf ()))
[Node (Leaf (1,())) (Leaf (2,())),
 Node (Leaf (2,())) (Leaf (1,()))]
```