Vorlesung Funktionale Programmierung, *WS 02/03*	*Dozentin: Dr. Ute Schmid,* *Tutor: Jens Waltermann*

Übungsblatt 6

(Ausgabe: Mo. 18.11.02, Abgabe: Fr. 22.11.02 12.00)

1. Aufgabe: Implementieren Sie die zu ARITH analoge structure:

```
structure polynom =
struct
type t = (int*real) list;
val zero = [];
fun sum ...
fun diff = ...
fun prod = ...
fun quo = ...
end;
```

   Mit dieser structure soll die Rechnung mit Polynomen implementiert werden. Die Repräsentation der Polynome durch Listen von `int * real`–Tupeln ist zu verstehen als Liste der Summanden: Der `int`–Wert gibt den jeweiligen Exponenten an, der `real`–Wert den Koeffizienten.

$$x^4 - \frac{1}{2}x^3 + 2 \qquad \Rightarrow \qquad [(4, 1.0), (3, \sim 0.5), (0, 2.0)]$$

   Die Koeffizienten sollen immer einen Wert $\neq 0$ haben.
   Die Polinomdivision soll das Ergebnis und den Rest als Tupel zurückgeben.

2. Aufgabe:

   Schreiben sie eine Funktion, die überprüft ob ein ihr übergebener Baum *höhenbalanciert* ist, d. h., ob sein linker Teilbaum sich in der Höhe um mehr als eine Ebene von dem rechten Teilbaum unterscheidet.

3. Aufgabe:

   Gegeben sei die signature:

```
signature ADT =
  sig
  type adt
  type element
  val empty:  adt -> bool
  val insert:  adt * element -> adt
  val delete:  adt * element -> adt
  val find:  adt * element -> bool
  end;
```

Implementieren Sie mit `structure avl: ADT` einen AVL-Baum mit den Eigenschaften, die aus Info A gut bekannt sein dürften. Nochmal zur Erinnerung: Ein AVL-Baum ist ein *Suchbaum*, in dem alle Knoten höhenbalanciert sind. Ein Suchbaum ist ein Baum, in dem für alle Knoten gilt daß sämtliche Schlüssel seines rechten Teilbaumes kleiner sind als der Schlüssel in der Wurzel und alle Schlüssel im linken Teilbaum grösser sind als der Schlüssel in der Wurzel (Keine Schlüssel doppelt!). Beim Einfügen und Löschen muß auf Ausgeglichenheit geprüft werden, und falls diese nicht mehr gegeben ist, müssen Rotationen durchgeführt werden. Beim Einfügen tritt höchstens eine Rotation auf, beim Löschen kann sich eine Rotation jedoch vom Blatt bis in die Wurzel fortpflanzen, so daß hier eine rekursive Formel erforderlich ist.

Wir geben vor:

```
datatype  (key,'a)avl = Lf|
          Br of (key*'a) * (key*'a)avl * (key*'a)avl;
```

In den Knoten sollen key/value-Paare abgespeichert werden ($\rightarrow$ `datatype elem`), wobei der key ein Typ sein muß, für den die Vergleichsoperatoren ($>, <, =$) definiert sind. Um die Überladung aufzulösen, sollten Sie eine Datei vorschalten die `key` beispielsweise auf `int` festsetzt.

4. Aufgabe (*): Gegeben seien die folgenden Funktionale auf Listen:

   - map f [] = [] | map f (x::l) = f(x)::(map f l);
   - filter p [] = [] | filter p (x::l) = if (p(x)) then x::(filter p l) else (filter p l);
   - reduce f [] e = e | reduce f (x::l) e = f(x, (reduce f l e));

   Beschreiben Sie, was diese Funktionen tun.
   Gibt es einen Zusammenhang zwischen diesen Funktionalen?
   Wie sehen diese Funktionale auf anderen Datenstrukturen aus (Bäume oder Graphen)?