TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen
Einführung in die Informatik 2
Prof. Dr. Helmut Seidl, Ralf Vogler, Stefan Schulze Frielinghaus

WS 2015/16
Übungsblatt 6

Aufgabe 6.1 [13 Punkte] **OCaml-Hausaufgabe: Listen und Records**

Zur Abgabe melden Sie sich mit Ihrer TUM-Kennung auf `https://vmnipkow3.in.tum.de` an und laden ihre `ha2.ml`-Datei (nicht umbenennen!) hoch. Die Ergebnisse der Tests sind sichtbar sobald die Abgabe abgearbeitet wurde. Die endgültige Bewertung erfolgt aber erst nach der Frist.

In der Datei gibt es drei Module mit Funktionen die zu implementieren sind:

a) `MyList`: mehr Funktionen auf polymorphen und Integer-Listen,

b) `NonEmptyList`: einige Funktionen für einen Listen-Typ der keine leeren Listen erlaubt,

c) `Db`: nutzen Sie Ihre oben definierten Funktionen um Berechnungen auf Listen von Records durchzuführen.

Aufgabe 6.2 **OCaml-Tutoraufgabe: Summentypen**

a) OCaml hat `let` und `let rec` — was sind die Unterschiede? In Haskell sind `let` immer rekursiv — was ist dadurch nicht mehr möglich?

b) Implementieren Sie folgende Funktionen. Welchen Vorteil hat das Einführen neuer Typen?

```
type n = float
type k = K of n (* SI: Kelvin *)
type c = C of n (* Celsius *)
let c_to_k : c -> k = ??
let k_to_c : k -> c = ??
```

c) Implementieren Sie für folgende Typen eine Funktion zum Auswerten von arithmetischen Ausdrücken (mit m können Variablen Werten zugewiesen werden):

```
type c = int (* only integer constants *)
and v = string (* variables *)
and m = v -> c (* environment mapping variables to values *)
and unop = Neg  (* unary operations *)
and binop = Add | Sub | Mul | Div (* binary operations *)
and e = C of c | V of v | Unop of unop*e | Binop of binop*e*e (* expressions *)

(* evaluate expressions *)
let rec eval_e : m -> e -> c = ??
```

d) Wir erweitern das vorherige Schema nun um Anweisungen (Zuweisungen und Ausgabe von Variablen). Implementieren Sie nun Funktionen zum Auswerten von Anweisungen und ganzen Programmen (Liste von Anweisungen).

```
type c ...
and s = Asn of v*e | Print of v (* statements: assignment, print variable *)
and p = s list (* a program is a list of statements *)

(* evaluate a statement and return environment *)
let eval_s : m -> s -> m = ??

(* evaluate a program (with side-effects) and return environment *)
let eval_p : m -> p -> m = ??
```