**Ludwig-Maximilians-Universität München** WS 2005/2006
**Institut für Informatik** Blatt 10
Prof. Dr. F. Kröger, Dr. M. Hölzl, F. Hacklinger

# Übungen zu Informatik I (Lösungsvorschlag)

**Aufgabe 10-1** **Kassensystem** (keine Abgabe)

```
(* Teilaufgabe (a) *)

signature TABELLESig =
sig
    type tabelle
    exception nicht_gefunden
    exception bereits_enthalten
    val neue_tabelle : (string * string * real) list -> tabelle
    val einfuegen    : (string * string * real) * tabelle -> tabelle
    val loeschen     : string * tabelle -> tabelle
    val aendern      : string * real * tabelle -> tabelle
    val bezeichnung  : string * tabelle -> string
    val preis        : string * tabelle -> real
end;;

(* Teilaufgabe (b) *)

structure TABELLE : TABELLESig =
struct
datatype tabelle = Tabelle of (string * string * real) list
exception nicht_gefunden
exception bereits_enthalten

fun neue_tabelle xs = Tabelle(xs)
fun einfuegen ((code, bez, preis), Tabelle xs) =
    let fun e [] = [(code, bez, preis)]
          | e ((c, b, p)::xs) = if c = code then raise bereits_enthalten
                                else (c, b, p)::(e xs);
    in Tabelle(e xs) end

fun loeschen (code, Tabelle xs) =
    let fun l [] = raise nicht_gefunden
          | l ((c, b, p)::xs) = if code = c
                        then xs
                        else (c, b, p)::(l xs)
    in Tabelle(l xs) end

fun find (code, Tabelle xs) =
    let fun f [] = raise nicht_gefunden
          | f ((c, b, p)::xs) = if code = c
```

```
                              then (c,b,p)
                              else f xs
    in f xs end

fun aendern(code, neuer_preis, Tabelle xs) =
    let fun a [] = raise nicht_gefunden
  | a ((c, b, p)::xs) = if c = code then (c, b, neuer_preis)::xs
    else (c, b, p)::(a xs)
    in Tabelle(a xs) end

fun bezeichnung (code, tab) = #2(find(code, tab))

fun preis (code, tab) = #3(find(code, tab))

end;;

open TABELLE;;

val t1 = neue_tabelle[("123","Zahnpasta",2.39),
                      ("0815","Leberwurst",1.99),
                      ("836","Cornflakes",3.49),
                      ("9234","Kleinwagen",19999.99)];

(* Teilaufgabe (c) *)

signature VERKAUFSig =
sig
type verkauf
val neuer_verkauf : verkauf
val neuer_artikel : int * string * verkauf -> verkauf
val artikelliste  : verkauf * tabelle -> (string * real * int * real) list
val gesamtpreis   : verkauf * tabelle -> real
end;;

(* Teilaufgabe (d) *)

structure VERKAUF : VERKAUFSig =
struct
datatype verkauf = Verkauf of (int * string) list
val neuer_verkauf = Verkauf[]

fun neuer_artikel(anzahl, code,Verkauf xs) =
    let fun na [] = [(anzahl, code)]
          | na ((n, c)::xs) = if c = code
                              then (n + anzahl, c)::xs
                              else (n, c)::(na xs)
    in Verkauf(na xs) end

fun artikelliste (Verkauf xs, tab) =
    map (fn (n, code) => (bezeichnung(code, tab), preis(code, tab),
```

```
                    n, real(n) * preis(code, tab)))
        xs
(* alternative Implementierung ohne map
    let fun art_liste [] = []
          | art_liste ((n, code)::xs) = (bezeichnung(code, tab), preis(code, tab),
n, real(n) * preis(code, tab))::art_liste(xs)
    in art_liste xs end *)

fun gesamtpreis (Verkauf xs, tab) =
    foldl (op +) 0.0 (map (fn (n, code) => real(n) * preis(code, tab)) xs)
(* alternative Implementierung ohne fold:
    let fun add [] = 0.0
          | add ((n, code)::xs) = real(n) * preis(code, tab) + add(xs)
    in add xs end *)
end;;

open VERKAUF;;

val v1 = neuer_artikel(2,"123",neuer_artikel(3,"0815",neuer_verkauf));;
val v2 = neuer_artikel(4,"836",neuer_artikel(3,"123",v1));;
```

**Aufgabe 10-2** **Endliche Mengen** (5 Punkte)

```
use "set.sml";

open SET;

(* Teilaufgabe a - Zunächst definieren wir die Funktion subset *)

fun subset(a, b) = if isemptyset(a) then true
                   else
                       let
                           val c = any(a)
                       in
                           member(c, b) andalso subset(delete(c, a), b)
                       end;;


(* Teilaufgabe b - Mit Hilfe der Funktion subset läßt sich die
   Funktion seteq wie folgt definieren: *)

fun seteq(a, b) = subset(a, b) andalso subset(b, a);;

(* Es gibt aber auch eine wesentlich effizientere Lösung: *)

fun seteq(a,b) = if isemptyset(a) then isemptyset(b)
                 else
                     let
                         val c=any(a)
```

```
                    in
                        member(c, b) andalso seteq(delete(c,a),delete(c,b))
                    end;;

(* Teilaufgabe c *)

fun intersect(a, b) = if isemptyset(a) orelse isemptyset(b) then emptyset
                      else
                          let
                              val c=any(a)
                          in
                              if member(c, b) then
                                  insert(c, intersect(delete(c,a),b))
                              else intersect(delete(c,a),b)
                          end;;


(* Teilaufgabe d - für die Lösung benötigen wir die in der Vorlesung
   definierte Funktion, die die Vereinigung zweier Mengen berechnet *)

fun union(a,b) = if isemptyset a then b
                 else if isemptyset b then a
                 else
                     let
                         val c=any a
                     in
                         if member(c, b) then union(delete(c,a),b)
                         else insert(c, union(delete(c,a),b))
end;;

fun cartprod(a,b) = if isemptyset a orelse isemptyset b then emptyset
                    else
                        let
                            val c=any a
                            val d=any b
                        in
                            insert((c,d),
                                   union(cartprod(delete(c,a),b),
                                         cartprod(a, delete(d,b))))
                        end;;
```

**Aufgabe 10-3** **Schlangen** (7 Punkte)

```
(* Teilaufgabe a *)

signature QUEUESig =
sig
    type 'a queue
    val emptyqueue    : 'a queue
```

```
    val isemptyqueue  : 'a queue -> bool
    val first         : 'a queue -> 'a
    val dequeue       : 'a queue -> 'a queue
    val enter         : 'a * 'a queue -> 'a queue
end;;

(* Teilaufgabe b *)

structure QUEUE : QUEUESig =
struct
type 'a queue = 'a list
val emptyqueue = nil
fun isemptyqueue l = null(l)
fun enter(a, q) = if isemptyqueue(q) then [a]
                    else hd(q)::enter(a, tl(q))
fun dequeue l = tl l
fun first l = hd l
end;;

open QUEUE;;

val q1 = enter(4,enter(3,enter(2,enter(1,emptyqueue))));;
val q2 = enter(~3,enter(7,enter(~5,q1)));;

(* Teilaufgabe c *)

fun posqueue q =
    let fun map q acc = if isemptyqueue q then acc
                        else let val h = first q
                             in map (dequeue q)
                                    (if h >= 0
                                     then enter(h, acc)
                                     else acc) end
    in map q emptyqueue end;;

(* alternative Implementierung von posqueue ohne map *)

fun posqueueemb(q, result) =
    if isemptyqueue(q) then result
    else if first(q) < 0 then posqueueemb(dequeue(q), result)
    else posqueueemb(dequeue(q), enter(first(q), result));

fun posqueue_alt q =
    posqueueemb(q, emptyqueue);

(* Teilaufgabe d *)

fun revqueue q = if isemptyqueue(q) then q
                 else enter(first(q), revqueue(dequeue(q)));;
```