TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen
A. Berlea, F. Forster, T. Gawlitza

SoS 2007
Lösungsvorschläge zu Blatt 6
Ausgegeben am: 22. Juni 2007
Abgabe bis: **06. Juli 2007**

Übungen zu Einführung in die Informatik II

Aufgabe 1 Datenstruktur (Lösungsvorschlag)

```
open List
open String

let empty_list = (0,[],0,[])
let is_empty = function
    (0,[],0,[]) -> true
  | _ -> false
let is_first = function
    (0,[],_,e::_) -> true
  | (_,_::_,_,_) -> false
let is_last = function
    (_,_,1,[e]) -> true
  | (_,_,_,_::_) -> false
let add_before (xc,xs,yc,ys) x = (xc,xs,yc+1,x::ys)
let add_behind l x =
    match l with
        (xc,xs,yc,e::ys) -> (xc+1,e::xs,yc,x::ys)
      | (xc,xs,0,[]) -> (xc,xs,1,[x])
let add = add_behind
let to_string str_of_e = function
    (_,xs,_,e::ys) ->
        let xs = map str_of_e (rev xs) in
        let xs = concat "," xs in
        let ys = map str_of_e ys in
        let ys = concat "," ys in
        "[" ^ xs ^ ",(" ^ (str_of_e e) ^ ")," ^ ys ^ "]"
  | (0,[],0,[]) -> "[]"
let len (xc,_,yc,_) = xc + yc
let next (xc,xs,yc,e::ys) = (xc+1,e::xs,yc-1,ys)
let prev (xc,x::xs,yc,ys) = (xc-1,xs,yc+1,x::ys)
let rec move l i =
    let (xc,xs,yc,ys) = l in
    if i = xc then
        l
```

```
    else if i > xc then
        move (next l) i
    else
        move (prev l) i
let ( ++ ) = add

let l = empty_list ++ "Das" ++ "ist" ++ "ein" ++ "Test"
let l = move l 1
let _ = print_string (to_string (fun x -> x) l)
let _ = print_string "\n"
```

Aufgabe 2 Powerpoint (Lösungsvorschlag)

```
type element =  Table of paragraph list list | Image of string |
                Link of string | Word of string | Itemize of paragraph list
and paragraph = Par of element list


type slide = {head:paragraph list option;
              foot:paragraph list option;
              title:string;
              content:paragraph list}

type presentation = slide list

let getCountMap presentation =
    let table = Hashtbl.create(100) in
    let doWord w =
      try
        let c = Hashtbl.find table w in
        Hashtbl.replace table w (c+1)
      with Not_found -> Hashtbl.replace table w 1 in

    let rec doPar (Par es) =  List.iter doElement es
    and doElement e =
            match e with
                Word w -> doWord w
              | Table rows -> List.iter (List.iter doPar) rows
           | Itemize pars -> List.iter doPar pars
              | _ -> () in
    let _ = List.iter (function slide ->
                      let _ = match slide.head with
                    Some pars -> List.iter doPar pars
                      | None -> () in
                      let _ = match slide.foot with
                    Some pars -> List.iter doPar pars
                      | None -> () in
                      let _ = doWord slide.title in
                      let _ = List.iter doPar slide.content in
                      ()
                        )
            presentation
    in
        table

let p = [{head = Some [Par [Word "this";Word "is";Word "the";Word "header"]];
          foot = None;
          title ="First";
          content = [Par [Word "this";Word "is";Word "the";Word "content"];
```

```
            Par [Word "An";Word "Itemize";Word "Environment";
                 Itemize [Par [Word "first"];
                          Par [Word "second"];
                          Par [Word "and";Word "third"]
                 ]
                ]
          ]

         };
         {head = None;
         foot = Some [Par [Word "a";Word "small";Word "footer"]];
         title ="Second";
          content = [Par [Word "The";Word "Great";Word "Wall"];
                     Par [Table [[Par [Word "He";Word "who";Word "has"];
                                  Par [Word "not";Word "climbed";Word "the"]];
                                 [Par [Word "Great";Word "Wall";Word "is"];
                                  Par [Word "not";Word "a";Word "true";Word "man"]]
                                ]
                          ]
                    ]
         }
        ]

let printHashMap t (fromName,from2String) (toName,to2String) =
    let _ = print_string (fromName^"\t\t\t\t\t"^toName^"\n") in
    let _ = print_string "-------------------------------------------------------\n" in
        Hashtbl.iter
        (function w -> function f ->
          print_string ((from2String w)^"\t\t\t\t\t"^(to2String f)^"\n"))
      t


let getFrequency p =
    let countMap = getCountMap p in
    printHashMap countMap ("Word",fun x -> x) ("Frequency",string_of_int)


let number presentation =
    let rec doit p curr =
      match p with
        [] -> []
      | slide::rest ->
          { head = (match slide.head with
                      Some pars -> Some ((Par [Word (string_of_int curr)])::pars)
                    | None -> Some [Par [Word (string_of_int curr)]]);
            foot = slide.foot;
            title = slide.title;
            content =
            slide.content}::
          (doit rest (curr+1))
    in doit presentation 1

let index presentation =
  let table = Hashtbl.create(100) in
  let rec doit p curr =
    match p with
      [] -> table
    | slide::rest ->
    let doWord w =
      try
            let l = Hashtbl.find table w in
```

```
            Hashtbl.replace table w (curr::l)
      with Not_found -> Hashtbl.replace table w (curr::[]) in

    let rec doPar (Par es) =  List.iter doElement es
    and doElement e =
          match e with
            Word w -> doWord w
          | Table rows -> List.iter (List.iter doPar) rows
          | _ -> () in

        let _ = match slide.head with
      Some pars -> List.iter doPar pars
    | None -> () in
    let _ = match slide.foot with
      Some pars -> List.iter doPar pars
    |   None -> () in
    let _ = doWord slide.title in
    let _ = List.iter doPar slide.content in
    doit rest (curr+1)
  in doit p 1

let list2String el2String l =
  let rec doit l =
    match l with [] -> ""
    |  h::r -> (el2String h)^(match r with h1::r1 -> "," | _ -> "")^(doit r)
  in "["^(doit l)^"]"

let getIndex p =
    let countMap = index p in
    printHashMap countMap ("Word",fun x -> x) ("Occurences",list2String string_of_int)
```