TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

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

SoS 2007
Lösungsvorschläge zu Blatt 5
Ausgegeben am: 8. Juni 2007
Abgabe bis: **22. Juni 2007**

Übungen zu Einführung in die Informatik II

Aufgabe 1 Terminkalender (Lösungsvorschlag)

```
type tag = Montag | Dienstag | Mittwoch | Donnerstag | Freitag | Samstag | Sonntag
type termin = { tag : tag; ort: string; beschreibung : string}
let terminkalender terminliste =
  let rec terminkalender2 (terminliste, mon, die, mit, don, fre, sam, son) =
    match terminliste with
      x::xs ->
    (match x.tag with
      Montag -> terminkalender2 (xs, x::mon, die, mit, don, fre, sam, son) |
      Dienstag -> terminkalender2 (xs, mon, x::die, mit, don, fre, sam, son) |
      Mittwoch  -> terminkalender2 (xs, mon, die, x::mit, don, fre, sam, son) |
      Donnerstag -> terminkalender2 (xs, mon, die, mit, x::don, fre, sam, son) |
      Freitag -> terminkalender2 (xs, mon, die, mit, don, x::fre, sam, son) |
      Samstag -> terminkalender2 (xs, mon, die, mit, don, fre, x::sam, son) |
      Sonntag -> terminkalender2 (xs, mon, die, mit, don, fre, sam, x::son))
    | [] -> mon@die@mit@don@fre@sam@son
  in terminkalender2 (terminliste, [],[],[],[],[],[],[])
```

Aufgabe 2 (H) Mini-Java-Interpreter (15 Punkte)

```
type variable = X | Y | Z

and expr =
    | Const of int
    | Var   of variable
    | Add   of expr * expr
    | Sub   of expr * expr
    | Mul   of expr * expr
    | Div   of expr * expr

let rec eval rho = function
    Add (e1,e2) -> (eval rho e1) + (eval rho e2) |
    Sub (e1,e2) -> (eval rho e1) - (eval rho e2) |
    Mul (e1,e2) -> (eval rho e1) * (eval rho e2) |
    Div (e1,e2) -> (eval rho e1) / (eval rho e2) |
    Const i     -> i |
    Var v       -> rho v
```

```
let update rho x new_val =
  match x with
      X -> (function X -> new_val | Y -> rho Y | Z -> rho Z)
    | Y -> (function X -> rho X | Y -> new_val | Z -> rho Z)
    | Z -> (function X -> rho X | Y -> rho Y | Z -> new_val)

type bool_expr =
    Not of bool_expr
  | And of bool_expr * bool_expr
  | Eq of expr * expr
  | Lt of expr * expr

let rec eval_bool rho = function
    Not b -> not (eval_bool rho b)
  | And(a,b) -> (eval_bool rho a) && (eval_bool rho b)
  | Eq(a,b) -> (eval rho a) = (eval rho b)
  | Lt(a,b) -> (eval rho a) < (eval rho b)

type stmt =
    Assign of variable * expr
  | Read of variable
  | Print of expr
  | If of bool_expr * stmt list * stmt list
  | While of bool_expr * stmt list

let rec run stmts rho =
  match stmts with
      [] -> rho
    | (Assign(x,expr))::stmts' ->
        let v = eval rho expr in
        run stmts' (update rho x v)
    | Read(x)::stmts' ->
        let _ = print_string "Zahl␣=␣" in
        let v = read_int () in
        run stmts' (update rho x v)
    | Print(expr)::stmts' ->
        let _ = print_string (string_of_int (eval rho expr)) in
        let _ = print_string "\n" in
        run stmts' rho
    | (If(b,if_stmts,else_stmts))::stmts' ->
        if eval_bool rho b then
            run (if_stmts @ stmts') rho
        else
            run (else_stmts @ stmts') rho
    | (While(b,body))::stmts' ->
        if eval_bool rho b then
            run (body @ stmts) rho
        else
            run stmts' rho

(** Initiale Variablen-Belegung **)
```

```
let rho_init = function X -> 0 | Y -> 0 | Z -> 0

(** Funktion zur Ausfuehrung ganzer Programme **)
let run prg = run prg rho_init

(** Ein Test Programm **)
let prg =
            [
                Read(X);
                Assign(Y,Const(1));
                Assign(Z,Const(0));
                While (Lt(Var(Y),Var(X)),
                [
                    Assign(Y,Add(Var(Y),Const(1)));
                    Assign(Z,Add(Var(Z),Var(Y)))
                ]);
                Print(Var(Z))
            ]

let _ = run prg
```