



Programmierung: Musterlösung zum 8. Übungsblatt

Prof. Gert Smolka und Thorsten Brunklaus

Aufgabe 8.1: Lexikographische Ordnung (10)

```
fun compare(nil , nil ) = EQUAL
  | compare(nil , ys ) = LESS
  | compare(xs , nil ) = GREATER
  | compare(x::xs, y::ys) = (case Char.compare(x,y) of
                             EQUAL => compare(xs,ys)
                             | x    => x)
```

Aufgabe 8.2: Dreiecke (5+5+5)

```
fun for n p =
  let
    fun for' i n p = if i > n then () else (p i ; for' (i+1) n p)
  in
    for' 1 n p
  end

fun triangle n =
  for n
  (fn k => (for (n-k+1)
               (fn _ => print "*") ; print "\ n"))

fun main s =
  triangle(valOf(Int.fromString s))

val _ =
  List.app main (CommandLine.arguments())
  handle _ => print "Arguments must be numbers \ n"
```

Aufgabe 8.3: Zeilennummern (10+5)

```

fun copy (n, instr, outstr) =
  case TextIO.inputLine instr of
    "" => ()
  | s  => (TextIO.output(outstr, Int.toString n ^ " : " ^ s) ;
          copy(n+1, instr, outstr) )

fun main(s, os) =
  let
    val instr  = TextIO.openIn s
    val outstr = (case os of
                    NONE      => TextIO.stdOut
                  | SOME s' => TextIO.openOut s')
  in
    copy(1, instr, outstr) ;
    TextIO.closeIn instr ;
    case os of NONE => () | _ => TextIO.closeOut outstr
  end

val _ =
  case CommandLine.arguments() of
    [s]      => main(s, NONE)
  | [s,s'] => main(s, SOME s')
  | _      => print "Need one or two files \ n"

```

Aufgabe 8.4: Taschenrechner (5+10+5+10+10+5+5+10)

```

fun num x nil      = (x, nil)
  | num x (c::cr) = if Char.isDigit c
                    then num (10*x + Char.ord c - Char.ord #"0") cr
                    else (x, c::cr)

datatype token = INT of int | ADD | SUB | MUL | LPAR | RPAR

exception Error of string

fun lex ts      nil      = rev ts
  | lex ts (#" " ::cs) = lex ts cs
  | lex ts (#"\ n"::cs) = lex ts cs
  | lex ts (#"+" ::cs) = lex (ADD::ts) cs
  | lex ts (#"-" ::cs) = lex (SUB::ts) cs
  | lex ts (#"*" ::cs) = lex (MUL::ts) cs
  | lex ts (#"(" ::cs) = lex (LPAR::ts) cs
  | lex ts (#")" ::cs) = lex (RPAR::ts) cs
  | lex ts ( c   ::cs) = if Char.isDigit c
                        then let val (x,cr) = num 0 (c::cs)
                              in lex (INT x::ts) cr
                              end
                        else raise Error("lex")

fun continue x f g h ts = let val (x',ts') = g ts
                          in h(f(x,x'), ts')

fun match t ts = if null ts orelse hd ts <> t
                 then raise Error("match") else tl ts

fun atom (INT x :: ts) = (x,ts)
  | atom (LPAR :: ts) = let val (x,tr) = exp ts
                        in (x, match RPAR tr)
                        end
  | atom _           = raise Error("atom")

and term' (x, MUL::ts) = continue x op* atom term' ts
  | term' (x,   ts ) = (x,ts)

and term ts          = term'(atom ts)

and exp' (x, ADD::ts) = continue x op+ term exp' ts
  | exp' (x, SUB::ts) = continue x op- term exp' ts
  | exp' (x,   ts ) = (x,ts)

and exp ts          = exp'(term ts)

fun eval s = let
    val (x, ts) = exp(lex nil (explode s))
  in
    if null ts then Int.toString x
    else raise Error("eval")
  end
handle
  Error(s) => "! Syntax error:" ^ s ^ "\ n"
  | Overflow => "! Overflow"

fun main () = (print "# " ;
               case TextIO.inputLine TextIO.stdIn of
                 "" => ()
                 | s => (print(eval s ^ "\ n") ; main()))

val _ = main()

```