Programmierung WS 2002 / 03:
Musterlösung zum 8. Übungsblatt

Prof. Dr. Gert Smolka, Dipl.-Inform. 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 ($15 = 3 * 5$)

```
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 k
              (fn _ => print "*") ; print "\n"))

(* Aufgabenteil b *)
fun main [s] = triangle(valOf(Int.fromString s))
  | main  _  = raise Subscript

val _ = main (CommandLine.arguments())
    handle _ => print "Argument must be a single number \n"

(* Aufgabenteil c *)
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 $(15 = 10 + 5)$

```
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 $(60 = 10 + 10 + 5 + 10 + 10 + 5 + 5 + 5)$

```
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

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
```

```
fun atom (INT x :: ts) = (x,ts)
  | atom (LPAR::ts)    = let val (x,tr) = exp ts
                         in case tr of
                             RPAR::tr' => (x,tr')
                           | _ => raise Error
                         end
  | atom      _        = raise Error

and term' (x, MUL::ts) = let val (x',tr) = atom ts
                         in term'(x*x', tr)
                         end
  | term' (x,    ts  ) = (x,ts)

and term  ts           = term'(atom ts)

and exp' (x, ADD::ts)  = let val (x',tr) = term ts
                         in exp'(x+x', tr)
                         end
  | exp' (x, SUB::ts)  = let val (x',tr) = term ts
                         in exp'(x-x', tr)
                         end
  | 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
             end
             handle
                 Error => "! Syntax error"
               | Overflow => "! Overflow"

fun main () = (print "# " ;
               print(eval(TextIO.inputLine TextIO.stdIn)) ;
               print "\n" ;
               main())

val _ = main()
```