```
(* $Date: 2005/11/09 20:34:44 $ *)

(* The stupid way of computing the Fibonnaci numbers *)
(* ------------------------------------------------ *)

fun fibS n = if n<=1 then 1 else (fibS (n-1))+(fibS (n-2));

(* The naive way of computing the Fibonacci numbers *)
(* ------------------------------------------------ *)
(* ...and the best for all practical purposes       *)

fun fibAux n a b = if n=0 then b else fibAux (n-1) b (a+b);
fun fibN   n     = if n<=1 then 1 else fibAux (n-1) 1 1;

(* The linear Algebra way of computing the Fibonacci numbers *)
(* --------------------------------------------------------- *)

(* we only have 2x2 Matrices, so do it more hands on *)

fun product ((a,b),(c,d)) ((e,f),(g,h)) = ((a*e+b*g,a*g+b*h),(c*e+d*g,c*f+d*h));
fun square A = product A A;
fun even n = (n mod 2 = 0);
fun power A n =  if n = 1 then A else
                 if even n then square (power A (n div 2))
                 else product (power A (n-1)) A;

val A = ((0,1),(1,1));

fun fibLAaux ((a,b),(_,_)) = a*1+b*1;

fun fibLA n = fibLAaux (power A n);
```

```
(* Matrices can be seen as mappings from indices to numbers *)
(* ------------------------------------------------------- *)
(* $Date: 2005/10/27 11:56:13 $ *)

val delta = (fn (i,j)=> if (i=j) then 1 else 0);
fun E k l = (fn (i,j) => delta(i,k) * (delta(j,l)));

(* Lists have a better print behaviour *)
fun index 0 = []
  | index n = (index (n-1))@[n];
fun nth 1 (x::xs) = x
  | nth n (x::xs) = nth (n-1) xs;

fun toList n m A = map (fn i => map (fn j => A(i,j)) (index m)) (index n);
fun fromList l = (fn (i,j) => nth j (nth i l));

(* some stupid examples *)

val telephone = fromList [[1,2,3],[4,5,6],[7,8,9]];
val iplustwoj = (fn(i,j) => i+2*j);

(* Some trivial operations on matrices *)
(* ----------------------------------- *)

(* \sum_{i=1}^n f(i) *)
fun sum n f = foldl (op +) 0 (map f (index n));


(* sum of two matrices *)
fun MatrixSum A B = (fn (i,j) => A(i,j) + (B(i,j)));

(* skalar times Matrix *)
fun MatrixScalar alpha A = (fn (i,j) => alpha * (A(i,j)));

(* Product of two matrices; needs to know the joint dimension,
   i.e., the n, if A is an k*n and B an n*m matrix *)
fun MatrixProduct A n B = (fn (i,k) => sum n (fn j => A(i,j)*(B(j,k))));

(* transpose of a matrix *)
fun Transpose A = (fn (i,j) => A(j,i));

(* ------------------------------------------------------------------

some examples...

> toList 3 3 iplustwoj;
val it = [[3, 5, 7], [4, 6, 8], [5, 7, 9]] : int list list
> toList 3 3 (E 1 2);
val it = [[0, 1, 0], [0, 0, 0], [0, 0, 0]] : int list list
> toList 3 3 (MatrixProduct (E 1 2) 3 iplustwoj);
val it = [[4, 6, 8], [0, 0, 0], [0, 0, 0]] : int list list
> toList 3 3 (MatrixProduct (E 2 2) 3 iplustwoj);
val it = [[0, 0, 0], [4, 6, 8], [0, 0, 0]] : int list list
> toList 3 3 (MatrixProduct (E 3 2) 3 iplustwoj);
val it = [[0, 0, 0], [0, 0, 0], [4, 6, 8]] : int list list
> toList 3 3 (MatrixProduct (E 3 1) 3 iplustwoj);
val it = [[0, 0, 0], [0, 0, 0], [3, 5, 7]] : int list list
> toList 2 2 (MatrixProduct (E 1 1) 2 (E 1 2));
val it = [[0, 1], [0, 0]] : int list list
> toList 2 2 (MatrixProduct (E 1 2) 2 (E 1 1));
val it = [[0, 0], [0, 0]] : int list list
> toList 3 3 (MatrixSum telephone (MatrixScalar 100 (E 1 2)));
val it = [[1, 102, 3], [4, 5, 6], [7, 8, 9]] : int list list
```

```
---------------------------------------------------------------------- *)
```