

```
(* Weekly exercises in INF3110/4110 week 38 15-19.09.2008 *)
(* ML-programming *)
```

```
(* Exercise 1
```

```
Write a function
```

```
  fun reverse (l: 'a list) : 'a list = ...
that reverses a list.
```

```
Ex: reverse(["A", "B", "C"]) gives ["C", "B", "A"]
    and reverse([1, 2, 3, 4]) gives [4, 3, 2, 1]
```

```
*)
```

```
fun reverse (nil)    = nil
  | reverse (s::ss) = reverse(ss)@[s];
```

```
reverse(["A", "B", "C"]);
reverse([1, 2, 3, 4]);
```

```
(* Hale-rekursiv versjon *)
```

```
fun reverse(ss) =
  let fun reverse (nil,res) = res
      | reverse (s::ss,res) = reverse(ss,s::res)
  in
    reverse(ss,nil)
  end;
```

```
reverse(["A", "B", "C"]);
reverse([1, 2, 3, 4]);
```

```
(* Exercise 2
```

```
Write the function
```

```
  fun rep(v, n) = ...;
that creates a list with n occurrences of v. The function should
work for arguments of any type.
```

```
Example:
```

```
- rep (17, 4);
val it = [17,17,17,17] : int list
- rep ("a", 5);
val it = ["a","a","a","a","a"] : string list
- rep ([1,2,3], 3);
val it = [[1,2,3],[1,2,3],[1,2,3]] : int list list
```

```
*)
```

```
fun rep (_,0) = []
  | rep (n,t) = n::rep(n,t-1);
```

```
rep (17, 4);
rep ("a", 5);
rep ([1,2,3], 3);
```

(\* **Exercise 3**

Write the function

```
fun mulall(v, l) = ...;
```

that multiplies all elements in *l* with the integer *v*.

NB! Your solution should use the *map*-function!

Example:

```
- mulall(~1, [2, 4, ~7, 10]);
```

```
val it = [-2,~4,7,~10] : int list
```

\*)

```
fun mulall (value,list) = map (fn(element) => element*value) list;
```

```
mulall(~1,[2,4,~7,10]);
```

(\* **Exercise 4**

Use the function *mulall* to define the function

```
fun powlist(x, n) = ...;
```

that returns a list with the exponentiations  $x^n$ ,  $x^{(n-1)}$ , ...  $x$ , 1 (where  $\wedge$  denotes "to the power of" ).

Example:

```
- powlist(3, 6);
```

```
val it = [729,243,81,27,9,3,1] : int list
```

\*)

(\* Løsning uten bruk av *mulall*: \*)

```
fun powlist (_, 0) = [1]
```

```
| powlist (x, n) = round(Math.pow(real(x),real(n))):powlist(x,n-1);
```

```
powlist(3,6);
```

(\* Løsning med bruk av *powlist*: \*)

```
fun powlist (x, 0) = [1]
```

```
| powlist (x, n) = mulall(x,powlist(x,n-1)) @ [1];
```

```
powlist(3,6);
```

(\* hale-rekursiv løsning, uten bruk av *mulall* \*)

```
fun powlist(x,n) =
```

```
  let fun powlist (x,0,res) = res
```

```
      | powlist (x,n,nil) = nil (* should not happen *)
```

```
      | powlist (x,n,y::ys) = powlist(x,n-1,(x*y)::(y::ys))
```

```
  in
```

```
    powlist(x,n,[1])
```

```
  end;
```

(\* Exercise 5

We want to create a function `mullist`, that sums up the product of two lists (which we assume to be non-empty and equally long).

Example:

```
- mullist([1, 0, ~1], [3, 4, 5]);  
val it = ~2 : int  
- mullist([1, 2, 3, 4], [1, 2, 3, 4]);  
val it = 30 : int  
(which is  $1*3 + 0*4 + ~1*5 = ~2$  og  $1*1 + 2*2 + 3*3 + 4*4 = 30$ ).
```

The exercise should be solved in the following manner:

a) Create a function `pair` which builds a list of tuples (pairs) from the two lists:

```
- pair([1, 0, ~1], [3, 4, 5]);  
val it = [(1,3),(0,4),(~1,5)] : (int * int) list  
- pair([1, 2, 3, 4], [1, 2, 3, 4]);  
val it = [(1,1),(2,2),(3,3),(4,4)] : (int * int) list  
b) Multiply the pairs by using map on the list.
```

c) Sum up the answer by using a `fold` on the list.

(See the "Note on functionals" for information about the fold functionals.)

\*)

```
fun pair(nil,_) = []  
  | pair(_,nil) = []  
  | pair(a::aa,b::bb) = (a,b)::pair(aa,bb);  
  
fun mullist(a,b) = foldr (op +)  
  0  
  (map (op* ) (pair(a,b)));
```

(\* Merk at "(op\* )" er lov, men ikke "(op\*)"! \*)

```
mullist([1,0,~1],[3,4,5]);  
mullist([1,2,3,4],[1,2,3,4]);
```

(\* Exercise 6

Assume that a polynomial is represented by a list such that  
[3, ~2, 0, 1] is the polynomial  $3x^3 - 2x^2 + 1$ .

Write the function

```
fun poly (x, p) = ... ;
```

that computes the polynomial  $p$  for the given  $x$ .

Example:

```
- val p = [3, ~2, 0, 1];  
val p = [3,~2,0,1] : int list  
- poly(2, p);  
val it = 17 : int  
- poly(~1, p);  
val it = ~4 : int
```

Hint: Use the functions `mullist` and `powlist` from previous exercises.

\*)

```
val p = [3,~2,0,1];          (* 3x^3 + -2*x^2 + 0*x^1 + 1*x^0)
```

```
fun poly (x,p) = mullist(powlist(x,length(p)-1),p);
```

```
poly(2,p);
```

```
poly(~1,p);
```

(\* Exercise 7

Given the following definition of the function `repeat`

```
fun repeat(f,d,l) =  
  case l of []      => d  
          | x :: l' => f(x,repeat(f,d,l'));
```

where  $d$  designates a "default"-value.

If we have a function:

```
fun minus (x:int, y:int) = x-y;
```

then

```
repeat(minus, 0, [1,2,3]);
```

will compute  $(1-(2-(3-0)))$  which gives the answer 2.

Write a `repeat`-like function that computes  $((1-2)-3)-0$  which should give the result -4. \*)

```
fun repeat(func,start,nil)      = start  
  | repeat(func,start,x::nil)  = func(x,start)  
  | repeat(func,start,x::y::list) = repeat(func,start,func(x,y)::list);
```

```
repeat((op -) , 10, [1,2,3]);
```