Tutorial 8

Week of November 7

1 Efficiency and let in ML

```
fun hundredth(x:real) =
    let
        val four = x * x * x * x;
        val twenty = four * four * four * four * four;
    in
        twenty * twenty * twenty * twenty * twenty
    end;


fun hundredth(x:real) =
    let
        val four = x * x * x * x
        and twenty = four * four * four * four * four;
    in
        twenty * twenty * twenty * twenty * twenty
    end;


fun median x =

    let fun medhelp ((x1::x2::xt),(y1::yt)) = medhelp(xt,yt)
          | medhelp (_,(median::_)) = median

          (* traverse twice as fast through one list *)
    in
       medhelp(x,x)
    end;

median [1,2,3];
median [1,2,3,4];
median [];


A bad idea:

fun sumcube n =
    let fun cube x = x * x * x
    in if n = 0 then 0 else cube n + sumcube (n-1)
    end;

cube is redefined at every level of recursion  :-(
```

Better idea:

```
fun sumcube n =
    let fun cube x = x * x * x;
        fun sumc 0 = 0
          | sumc m = cube m + sumc (m-1)
    in sumc n
    end;
```

2 Mutual Recursive Types

Want to represent a tree with arbitrary #of branches.

See the diagram first ...

Defining mutually recursive datatypes (using **and**).

```
1 -datatype tree = Empty| Node of int*forest
2      and forest= Nil  | Cons of  tree*forest
  datatype tree = Empty | Node of int * forest
  datatype forest = Cons of tree * forest | Nil

3 -val t1=Node(2,Nil);
  ??
4 -val t2=Node(3,Nil);
  ??
5 -val t3=Node(7,Cons(t1,Cons(t2,Nil)));
  ??
6 -val t4=Node(5,Nil);
  ??
7 -val t5=Node(1,Nil);
  ??
8 -val t6=Node(2,Cons(t5,Cons(t4,Cons(t3,Nil))));
  ??
```

We want to count how many nodes are in a tree.

solution: 1+ #of nodes in its subtrees (i.e. forest)

```
1 -fun numnodeT (Empty)=0
2     | numnodeT (Node(data,f))= 1+ numnodeF(f)
3   and
4       numnodeF(Nil) = 0
5      |numnodeF(Cons(t,f))= ???


   val numnodeT = fn : tree -> int
   val numnodeF = fn : forest -> int

(* Note that numnodeT and numnodeF are
   mutually recursive.*)

6  -numnodeT(t6)
   ??
```

3 Exceptions

```
(* two exceptions *)
exception Zero of int;
exception Negative of int;

(* posHarmonic = fn : int -> real
 * return Harmonic of n
 * Pre: n>=1, n is an integer
 *)
fun posHarmonic 1 = 1.0
|   posHarmonic n = 1.0/real(n) + posHarmonic(n-1);


(* harmonic = fn : int -> real
 * return Harmonic of n, for n >= 1
 * raise an exception Zero, for n=0
 * raise an exception Negative, for ow
 *)
fun harmonic n = if n >= 1 then posHarmonic n
                 else if n = 0 then raise Zero n
                 else raise Negative n;

- harmonic ~1;
uncaught exception Negative raised at: ...

- harmonic 0;
uncaught exception Zero raised at: ...

- harmonic 10;
val it = 2.92896825397 : real
```

```
(* harmonicList = fn : int list -> real list
 * Param: list L of integers
 * Return:
 *  a list of harmonic(element) for each element in L
 *  handle all exceptions by inserting 0.0 in place of the
 *  number which caused an Zero exception, and ~1.0 in place
 * of the number which caused a Negative exception and printing
 * an appropriate message.
 *)
fun harmonicList [] = []
|   harmonicList (n::rest) =
        harmonic(n)::harmonicList(rest)
        handle Zero(X) => (print (Int.toString(X)^" is zero\n");
                           0.0::harmonicList(rest))
        |      Negative(X) =>(print (Int.toString(X)^" is negative\n");
                              ~1.0::harmonicList(rest));


- harmonicList [1,~2,2];
~2 is negative
val it = [1.0,~1.0,1.5] : real list

- harmonicList [1,2,0];
0 is zero
val it = [1.0,1.5,0.0] : real list

- harmonicList [~5, 0, 5];
~5 is negative
0 is zero
val it = [~1.0,0.0,2.28333333333] : real list

- harmonicList [1,2,10];
val it = [1.0,1.5,2.92896825397] : real list
```