

Sample solutions for Homework 1

```
(* #1. write a function rotate(list:int list, n:int) that takes a list
    and an integer as input, and performs a circular shift of the
    elements in the list in anti-clockwise direction by n positions.
```

```
    For example: rotate([1,2,3,4,5],3); => [4,5,1,2,3].
```

```
                rotate([1,2,3,4,5],1); => [2,3,4,5,1].
```

```
    Your program should handle the case where n is larger
    than the length of the list.
```

```
    fun rotate(a : int list, n: int)
```

```
*)
```

```
print " -- problem 1 -- \n";
```

```
fun
```

```
  rotate([] : int list, n : int)=[] |
```

```
  rotate((x::a) : int list, n: int)=if n>0
```

```
    then rotate(a@(x::[]), n-1) else (x::a);
```

```
val p1=rotate;
```

```
val pla=([1,2,3,4,5],3);
```

```
p1 pla;
```

```
val plb=([1,2,3,4,5],1);
```

```
rotate plb;
```

```
val plc=([1,2,3,4,5],7);
```

```
p1 plc;
```

```
print " -- end of problem 1 -- \n -- beg of problem 2 -- \n";
```

```
(* problem #2 matrix multiplication two MxN and NxT matrix *)

fun find(h::t,1) = h |
  find(h::t,n) =(t, n-1);

fun matx_add([], [], n) = 0 |
  matx_add(h::t, a::b, n) = h*find(a,n) + matx_add(t,b,n);

fun multiply(a,b,n,0)=[] |
  multiply(a,l,n,m)= matx_add(a,l,n)::multiply(a,l,n+1,m-1);
fun len1([])=0 |
  len1(h::t)=1+len1(t);
fun len(h::t)=len1(h);

fun matrix_mult([],a)=[] |
  matrix_mult(h::t,a)=multiply(h,a,1,len(a))::matrix_mult(t,a);
```

```
print "-- beg of problem 3 -- \n";

(* 3. Program the Tower of Hanoi puzzle as discussed in class. *)
fun hanoi(n,a,b,c)=if n=0 then []
                  else hanoi(n-1,a,c,b)@[a,b]@hanoi(n-1,c,b,a);

val p3=hanoi;
val p3a=(3, "a", "b", "c");
p3 p3a;

print " -- end of problem 3 -- \n -- beg of problem 4 -- \n";
```

```
(* 4. List elements do not just have to be simple values --
   we can use tuples as list elements, for example.
   A list such as [(2,3.0),(5,4.4),(10,1.2)] is a 3-element list
   where each element is an int*real tuple
   (each list element must still be of the same type).
   In this problem, we want to calculate
   the total mileage, total gasoline used,
   and our average miles per gallon for the whole trip.
   The trip is composed of a number of segments.
   So, our input is a list, each element is the mileage and
   gasoline used for one segment of the trip --
   in an int*real tuple (the miles are just integers).
   You are to write a function mpg(trip: (int*real) list)
   that takes a list of miles*gallons tuples
   and returns an int*real*real tuple (a 3-tuple),
   of the total mileage, the total gasoline used,
   and the miles per gallon for the whole trip.
   You can (and should) define other functions to help you
   if you need to (hint: What are the first two values you have
   to return from this function?
   Perhaps you could write a function for each).
   [(2,3.0),(5,4.4),(10,1.2)] of [miles, gallons]
*)
fun tsumi(l:int list,s:int)=s |
  tsumi((x::t),s)=tsumi(t,x+s);
fun sumi(l:int list)=tsumi(l,0);
fun tsumr(l:(int*real) list,s:real)=s |
  tsumr((x::t),s)=tsumr(t,x+s);
fun sumr(l:(int*real) list)=tsumr(l,0.0);

fun zip(x::xs,y::ys)=(x,y)::zip(xs,ys)
  | zip _ = [];

(*
fun conspair((x,y),[xs,ys])=(x::xs,y::ys);
fun unzip[]=([],[])
  | unzip(pair::pairs)=conspair(pair, unzip pairs);
*)

fun unzip[]=([],[])
  | unzip((x,y)::pairs)=
    let val(xs,ys)=unzip pairs
    in (x::xs, y::ys) end;

fun rev_unzip([],xs,ys)=(xs,ys)
  | rev_unzip((x,y)::pairs,xs,ys)=rev_unzip(pairs,x::xs,y::ys);

fun prob4(list)=
  let val (xs,ys)=unzip list;
      val txs=sumi(xs);
      val tys=sumr(ys);
      val avg=tys/(real txs);
  in (txs, tys, avg) end;

val p4=prob4;
val p4a=([(2,3.0),(5,4.4),(10,1.2)]);
```

```
p4 p4a;
```

```
val p4b=([(2,3.0)]);
```

```
p4 p4b;
```

```
val p4c=([(10,3.0),(10,3.0),(10,3.0)]);
```

```
p4 p4c;
```

```
print " -- end of problem 4 -- \n -- beg of problem 5 -- \n";
```

```

(* 5. Suppose we represent sets (no duplicated elements) ranging
over integers as lists of integers.
Write ML functions for performing the following set operations:
  union(S1,S2): returns the set-union of sets S1 and S2.
  intersection(S1,S2): returns set-intersection of sets S1 and S2.
  setdiff(S1,S2): returns the set-difference of sets S1 and S2
  subset(S1,S2): returns true if S1 is a subset of set S2,
                 false otherwise.
*)

(* union(s1,s2) *)
fun union([],s2)=s2 |
  union(s1,[])=s1 |
  union(x::s1,s2)=if member(x,s2) then union(s1,s2)
                  else x::union(s1,s2);

(* fun intersection(s1,s2) *)
fun intersection([],s2)=[] |
  intersection(s1,[])=[] |
  intersection(x::s1,s2)=if member(x,s2) then x::intersection(s1,s2)
                          else intersection(s1,s2);

(* fun setdiff(s1,s2) *)
fun setdiff1([],s2)=[] |
  setdiff1(s1,[])=[] |
  setdiff1(x::s1,s2)=if member(x,s2) then setdiff1(s1,s2)
                     else x::setdiff1(s1,s2);

fun setdiff2([],s2)=[] |
  setdiff2(s1,[])=[] |
  setdiff2(s1,y::s2)=if member(y,s1) then setdiff2(s1,s2)
                     else y::setdiff2(s1,s2);

fun setdiff(s1,s2)=union(setdiff1(s1,s2),setdiff1(s2,s1));

(* subset(s1,s2) *)
fun subset([],[])=true |
  subset([],s2)=true |
  subset(s1,[])=false |
  subset(s1,s2)=if (s1=intersection(s1,s2)) then true else false;

print " p5 -- union -- \n";
val p5=union;
val p5a=([1,2,3],[1,2]);
p5 p5a;
val p5b=([1,2,3],[2,3,4]);
p5 p5b;
val p5c=([1,2],[3,4]);
p5 p5c;
val p5d=([1],[1,2]);
p5 p5d;

print " p5 -- intersection -- \n";
val p5=intersection;
val p5a=([1,2,3],[1,2]);
p5 p5a;

```

```
val p5b=([1,2,3],[2,3,4]);
p5 p5b;
val p5c=([1,2],[3,4]);
p5 p5c;
val p5d=([1],[1,2]);
p5 p5d;

print " p5 -- setdiff -- \n";
val p5=setdiff;
val p5a=([1,2,3],[1,2]);
p5 p5a;
val p5b=([1,2,3],[2,3,4]);
p5 p5b;
val p5c=([1,2],[3,4]);
p5 p5c;
val p5d=([1],[1,2]);
p5 p5d;

print " p5 -- subset -- \n";
val p5=subset;
val p5a=([1,2,3],[1,2]);
p5 p5a;
val p5b=([1,2,3],[2,3,4]);
p5 p5b;
val p5c=([1,2],[3,4]);
p5 p5c;
val p5d=([1],[1,2]);
p5 p5d;

print " -- end of problem 5 -- \n -- beg of problem 6 -- \n";
```

```

(* 6. Program QuickSort in ML *)
fun insert(x,[],)=[x] |
  insert(x,(y::t))=if x<y then (x::(y::t)) else (y::(insert(x,t)));
fun isort([],l)=l |
  isort((x::t),l)=isort(t,insert(x,l));
fun insertionsort(l)=isort(l,[]);

fun merge([],ys)=ys
  | merge(xs,[])=xs
  | merge(x::xs, y::ys)=
    if x<=y then x::merge(xs,y::ys)
    else y::merge(x::xs,ys);

(*
fun quicksort[]=[]
  | quicksort[x]=[x]
  | quicksort(a::bs)=
    let
      fun partition(left,right,[])=(quicksort left) @ (a :: quicksort right)
        | partition(left,right,x::xs)=
          if x<=a then partition(x::left,right,xs)
          else partition(left,x::right,xs)
    in partition([],[],bs);
*)

(* ----- QUICK SORT ----- *)
fun QuickSort R [] = []
  | QuickSort R (head::tail) =
    let fun partition R pivot [] = ([], [])
        | partition R pivot (front::rest) =
          let val (lesser, greater) = partition R pivot rest
            in if R (front, pivot) then (front::lesser, greater)
              else (lesser, front::greater)
            end
        val (Left, Right) = partition R head tail;
        val LeftSorted = QuickSort R Left;
        val RightSorted = QuickSort R Right;
    in LeftSorted @ (head::RightSorted)
    end;

val p6 = QuickSort;
val p6data=[~10, ~20, ~5, 10, 0, 12, 4, 1, 0, ~13];

print " p6 -- ascending order -- \n";
p6 (op <) [~10, ~20, ~5, 10, 0, 12, 4, 1, 0, ~13];
p6 (op <=) [~10, ~20, ~5, 10, 0, 12, 4, 1, 0, ~13];

print " p6 -- descending order -- \n";
p6 (op >) [~10, ~20, ~5, 10, 0, 12, 4, 1, 0, ~13];
p6 (op >=) [~10, ~20, ~5, 10, 0, 12, 4, 1, 0, ~13];

```

```
print " -- end of problem 6 -- \n -- beg of problem 7 -- \n";
```

```
(* 7. Program bubblesort in ML *)

fun bubble R [] = []
  | bubble R [h] = [h]
  | bubble R (x1::x2::xs) = if R (x1, x2) then x1::(bubble R (x2::xs))
                           else x2::(bubble R (x1::xs));

fun unsorted R [] = false
  | unsorted R [h] = false
  | unsorted R (x1::x2::xs) =
    if (x1=x2) then (unsorted R (x2::xs))
    else if R (x1, x2) then (unsorted R (x2::xs))
    else true;

fun bubbleSort R L = if (unsorted R L)
                    then (bubbleSort R (bubble R L)) else L;

val p7 = bubbleSort;
val p7data=[~10, ~20, ~5, 10, 0, 12, 4, 1, 0, ~13];

print " p7 -- ascending order -- \n";
p7 (op <)  [~10, ~20, ~5, 10, 0, 12, 4, 1, 0, ~13];
p7 (op <=) [~10, ~20, ~5, 10, 0, 12, 4, 1, 0, ~13];

print " p7 -- descending order -- \n";
p7 (op >)  [~10, ~20, ~5, 10, 0, 12, 4, 1, 0, ~13];
p7 (op >=) [~10, ~20, ~5, 10, 0, 12, 4, 1, 0, ~13];

print " -- end of problem 7 -- \n -- beg of problem 8 -- \n";
```

```
(* 8. Program the tail recursive versions of the reverse and
    sum functions discussed in class.
    Use the accumulator trick to write an O(n) function
    for computing fibonacci numbers.
```

```
*)
```

```
fun reversen([],p)=p |
    reversen((x::t),p)=reversen(t,(x::p));
fun reverse(l)=reversen(l,[]);
```

```
val p8r=reverse;
p8r [1, 2, 3, 4, 5];
```

```
fun sumn([],:int list,s:int)=s |
    sumn((x::t),s)=sumn(t,x+s);
fun sum(l:int list)=sumn(l,0);
```

```
val p8s=sum;
p8s [1, 2, 3, 4, 5];
```

```
print " -- end of problem 8 -- \n";
```

Standard ML of New Jersey v110.60 [built: Fri Nov 10 15:18:21 2006]
- use "hw1.ml";
[opening hw1.ml]

```
-- problem 1 --  
val it = () : unit  
val rotate = fn : int list * int -> int list  
val p1 = fn : int list * int -> int list  
val p1a = ([1,2,3,4,5],3) : int list * int  
val it = [4,5,1,2,3] : int list  
val p1b = ([1,2,3,4,5],1) : int list * int  
val it = [2,3,4,5,1] : int list  
val p1c = ([1,2,3,4,5],7) : int list * int  
val it = [3,4,5,1,2] : int list  
-- end of problem 1 --
```

```
-- beg of problem 3 --  
val it = () : unit  
val hanoi = fn : int * 'a * 'a * 'a -> ('a * 'a) list  
val p3 = fn : int * 'a * 'a * 'a -> ('a * 'a) list  
val p3a = (3,"a","b","c") : int * string * string * string  
val it =  
  [("a","b"),("a","c"),("b","c"),("a","b"),("c","a"),("c","b"),("a","b")]  
  : (string * string) list  
-- end of problem 3 --
```

```
-- beg of problem 4 --
val it = () : unit
val tsumi = fn : int list * int -> int
val sumi = fn : int list -> int
val tsumr = fn : real list * real -> real
val sumr = fn : real list -> real
val zip = fn : 'a list * 'b list -> ('a * 'b) list
val unzip = fn : ('a * 'b) list -> 'a list * 'b list
val rev_unzip = fn : ('a * 'b) list * 'a list * 'b list -> 'a list * 'b list
val prob4 = fn : (int * real) list -> int * real * real
val p4 = fn : (int * real) list -> int * real * real
val p4a = [(2,3.0),(5,4.4),(10,1.2)] : (int * real) list
val it = (17,8.6,0.505882352941) : int * real * real
val p4b = [(2,3.0)] : (int * real) list
val it = (2,3.0,1.5) : int * real * real
val p4c = [(10,3.0),(10,3.0),(10,3.0)] : (int * real) list
val it = (30,9.0,0.3) : int * real * real
-- end of problem 4 --
```

```

-- beg of problem 5 --
val it = () : unit
val union = fn : "a list * "a list -> "a list
val intersection = fn : "a list * "a list -> "a list
val setdiff1 = fn : "a list * "a list -> "a list
val setdiff2 = fn : "a list * "a list -> "a list
val setdiff = fn : "a list * "a list -> "a list
hw1.ml:202.25 Warning: calling polyEqual
val subset = fn : "a list * "a list -> bool
p5 -- union --
val it = () : unit
val p5 = fn : "a list * "a list -> "a list
val p5a = ([1,2,3],[1,2]) : int list * int list
val it = [3,1,2] : int list
val p5b = ([1,2,3],[2,3,4]) : int list * int list
val it = [1,2,3,4] : int list
val p5c = ([1,2],[3,4]) : int list * int list
val it = [1,2,3,4] : int list
val p5d = ([1],[1,2]) : int list * int list
val it = [1,2] : int list
p5 -- intersection --
val it = () : unit
val p5 = fn : "a list * "a list -> "a list
val p5a = ([1,2,3],[1,2]) : int list * int list
val it = [1,2] : int list
val p5b = ([1,2,3],[2,3,4]) : int list * int list
val it = [2,3] : int list
val p5c = ([1,2],[3,4]) : int list * int list
val it = [] : int list
val p5d = ([1],[1,2]) : int list * int list
val it = [1] : int list
p5 -- setdiff --
val it = () : unit
val p5 = fn : "a list * "a list -> "a list
val p5a = ([1,2,3],[1,2]) : int list * int list
val it = [3] : int list
val p5b = ([1,2,3],[2,3,4]) : int list * int list
val it = [1,4] : int list
val p5c = ([1,2],[3,4]) : int list * int list
val it = [1,2,3,4] : int list
val p5d = ([1],[1,2]) : int list * int list
val it = [2] : int list
p5 -- subset --
val it = () : unit
val p5 = fn : "a list * "a list -> bool
val p5a = ([1,2,3],[1,2]) : int list * int list

```

```
val it = false : bool
val p5b = ([1,2,3],[2,3,4]) : int list * int list
val it = false : bool
val p5c = ([1,2],[3,4]) : int list * int list
val it = false : bool
val p5d = ([1],[1,2]) : int list * int list
val it = true : bool
-- end of problem 5 --
```

```
-- beg of problem 6 --
val it = () : unit
val insert = fn : int * int list -> int list
val isort = fn : int list * int list -> int list
val insertionsort = fn : int list -> int list
val merge = fn : int list * int list -> int list
val QuickSort = fn : ('a * 'a -> bool) -> 'a list -> 'a list
val p6 = fn : ('a * 'a -> bool) -> 'a list -> 'a list
val p6data = [~10,~20,~5,10,0,12,4,1,0,~13] : int list
p6 -- ascending order --
val it = () : unit
val it = [~20,~13,~10,~5,0,0,1,4,10,12] : int list
val it = [~20,~13,~10,~5,0,0,1,4,10,12] : int list
p6 -- descending order --
val it = () : unit
val it = [12,10,4,1,0,0,~5,~10,~13,~20] : int list
val it = [12,10,4,1,0,0,~5,~10,~13,~20] : int list
-- end of problem 6 --
```

```
-- beg of problem 7 --
val it = () : unit
val bubble = fn : ('a * 'a -> bool) -> 'a list -> 'a list
hw1.ml:327.15 Warning: calling polyEqual
val unsorted = fn : ("a * "a -> bool) -> "a list -> bool
val bubbleSort = fn : ("a * "a -> bool) -> "a list -> "a list
val p7 = fn : ("a * "a -> bool) -> "a list -> "a list
val p7data = [~10,~20,~5,10,0,12,4,1,0,~13] : int list
p7 -- ascending order --
val it = () : unit
val it = [~20,~13,~10,~5,0,0,1,4,10,12] : int list
val it = [~20,~13,~10,~5,0,0,1,4,10,12] : int list
p7 -- descending order --
val it = () : unit
val it = [12,10,4,1,0,0,~5,~10,~13,~20] : int list
val it = [12,10,4,1,0,0,~5,~10,~13,~20] : int list
-- end of problem 7 --
```

```
-- beg of problem 8 --  
val it = () : unit  
val reversen = fn : 'a list * 'a list -> 'a list  
val reverse = fn : 'a list -> 'a list  
val p8r = fn : 'a list -> 'a list  
val it = [5,4,3,2,1] : int list  
val sumn = fn : int list * int -> int  
val sum = fn : int list -> int  
val p8s = fn : int list -> int  
val it = 15 : int  
-- end of problem 8 --  
val it = () : unit  
val it = () : unit  
-
```