Module Title: Functional Programming and Specification SAMPLE EXAM
Exam Diet (Dec/April/Aug): April 2010

1. (a)

```
load "Int"

fun depth(empty) = 0
  | depth(tip _) = 1
  | depth(node(t,t')) =
        let val d = depth t and d' = depth t'
        in if d=0 andalso d'=0 then 0 else 1 + Int.max(d,d')
        end

fun tips(0,_) = []
  | tips(1,empty) = []
  | tips(1,tip x) = [x]
  | tips(1,node(_,_)) = []  (* can be omitted: it is subsumed by the next case *)
  | tips(n,node(t,t')) = tips(n-1,t) @ tips(n-1,t')
  | tips(n,_) = []

fun deepest t = tips(depth t, t)
```

(b)

```
(* depth as in part (a) *)

fun deepest empty = []
  | deepest(tip x) = [x]
  | deepest(node(t,t')) =
      let val dt = depth t and dt' = depth t' in
        if dt>dt' then deepest t
        else if dt<dt' then deepest t'
             else (deepest t)@(deepest t')
      end
```

(c) There are various solutions. The following one uses the idea in part (a) above.

```
load "Int"

fun shallowness(empty) = 1000000
                (* ideally the largest available integer, see Int.maxInt, but ML
                   doesn't guarantee that there is a maximum integer *)
  | shallowness(tip _) = 1
  | shallowness(node(t,t')) = 1 + Int.min(shallowness t, shallowness t')

(* tips as in part (a) *)

fun shallowest t = tips(shallowness t, t)
```

2. (a) (i).

```
val empty = []
```

(ii).

```
fun member(z,nil) = false
  | member(z,(a,b)::l) =
      if z<a then false
      else if z<=b then true
      else member(z,l)
```

(iii).

```
fun delete(z,nil) = nil
  | delete(z,(a,b)::l) =
      if z<a then (a,b)::l
      else if z=a then if z=b then l else (z+1,b)::l
      else if z<b then (a,z-1)::(z+1,b)::l
      else if z=b then (a,z-1)::l
      else (a,b)::delete(z,l)
```

(b)

```
signature ELEM =
  sig
    eqtype t
    val lt : t * t -> bool
    val succ : t -> t
    val pred : t -> t
  end

signature SET =
  sig
    type elem
    type set
    val empty : set
    val insert : elem * set -> set
    val member : elem * set -> bool
    val delete : elem * set -> set
  end

functor Interval(structure E:ELEM):>SET where type elem=E.t =
  struct
    type elem = E.t
    type set = (elem*elem) list

    val empty = []

    fun member(z,nil) = false
      | member(z,(a,b)::l) =
          if E.lt(z,a) then false
          else if E.lt(z,b) orelse z=b then true
          else member(z,l)

    fun delete(z,nil) = nil
      | delete(z,(a,b)::l) =
          if E.lt(z,a) then (a,b)::l
          else if z=a then if z=b then l else (E.succ z,b)::l
          else if E.lt(z,b) then (a,E.pred z)::(E.succ z,b)::l
          else if z=b then (a,E.pred z)::l
          else (a,b)::delete(z,l)

    local
      fun insert'(z,nil) = [(z,z)]
        | insert'(z,(a,b)::l) =
            if E.lt(z,a) then (z,z)::(a,b)::l
            else if E.lt(z,b) orelse z=b then (a,b)::l
```

```
            else (a,b)::insert'(z,l)
      fun fix nil = nil
        | fix [(a,b)] = [(a,b)]
        | fix((a,b)::(c,d)::l) =
            if (E.succ b)=c then fix((a,d)::l) else (a,b)::(fix((c,d)::l))
    in
      fun insert(z,l) = fix(insert'(z,l))
    end
  end
```

The only difference between transparent and opaque signature ascription for this example is that for transparent signature ascription the header could be simplified to

```
functor Interval(structure E:ELEM):SET = ...
```

3. (a)

```
fun ncompose(f,0) = (fn x => x)
  | ncompose(f,n) = f o ncompose(f,n-1)
```

(b)

```
fun lcompose flist = foldr (op o) (fn x=>x) flist
```

(c)

```
fun createGate(f,n,s,v) =
  let
    val password = ncompose(f,n) s
  in
    (fn pwd => if password=pwd then SOME v else NONE)
  end
```

(d)

```
local
  fun helper g (f,s,n) =
    case g(ncompose(f,n) s)
      of NONE => helper g (f,s,n+1)
       | SOME v => v
in
  fun crack g (f,s) = helper g (f,s,0)
end
```