```
//1.1
let inv ms =
    (ms = List.distinct ms) &&
    (List.forall (fun (e,n) -> n > 0) ms);;

//1.2
let rec insert e n = function
    | [] -> []
    | (x,y)::xs when x=e -> (x,y+n)::xs
    | (x,y)::xs -> (x,y)::(insert e n xs);;

//1.3

let rec numberOf e = function
    | [] -> 0
    | (x,y)::xs when x=e -> y
    | (x,y)::xs -> numberOf e xs;;

//1.4
let delete e ms = List.filter (fun (x,y) -> e <> x) ms;;

//1.5
let rec union ms1 ms2 =
    match (ms1, ms2) with
    | [], _ -> ms2
    | _, [] -> ms1
    | (x,y)::xs, ms2 -> let msnew = insert x y ms2
                        union xs msnew;;

//1.6
let inv2 ms = Map.forall (fun e n -> n > 0) ms;;

let insert2 e n ms =
    if Map.containsKey e ms then    let n' = Map.find e ms
                                    Map.add e (n+n') ms
    else Map.add e n ms;;

let delete2 e ms = Map.remove e ms;;

//2.1
//f int -> 'a list -> (int * 'a) list
//Makes a new of tuples (i[j], x[j]) where i[j] = i[j-1] * i[j-1]

//g ('a -> bool) -> 'a Tree _> 'a Tree option
//Finds the first element in the tree which satisfies p
//and returns the tree traversed so far

let rec f i = function
    | [] -> []
    | x::xs -> (i,x)::f (i*i) xs;;

type 'a Tree = | Lf
```

```
               | Br of 'a Tree * 'a * 'a Tree;;


f 3 [1;2;3;4;5]

let rec g p = function
    | Lf -> None
    | Br(_,a,t) when p a -> Some t
    | Br(t1,a,t2) ->    match g p t1 with
                        | None -> g p t2
                        | res -> res;;

//2.2.1 tail-recursive
let rec fA i a = function
    | [] -> List.rev a
    | x::xs -> fA (i*i) ((i,x)::a) xs;;

fA 3 [] [1;2;3;4;5]

//2.2.1 continuation-based
let rec fC i c = function
    | [] -> c []
    | x::xs -> fC (i*i) (fun v -> c((i,x)::v)) xs;;

fC 3 id [1;2;3;4;5]

//2.3
let rec h f (n,e) =
    match n with
    | 0 -> e
    | _ -> h f (n-1, f n e);;

let A = Seq.initInfinite id;;

let B = seq { for i in A do
                    for j in seq {0..i} do
                            yield (i,j)};;

let C = seq { for i in A do
                    for j in seq {0..i} do
                            yield (i-j,j)};;

let X = Seq.toList (Seq.take 4 A)
let Y = Seq.toList (Seq.take 6 B)
let Z = Seq.toList (Seq.take 10 C)

h (*) (4,2)

//2.3
//h (*) (4,1) = (4*3*2*1)*(1) = 24
//h (*) (4,1) = (4*3*2*1)*(2) = 48
//h (int -> 'a -> 'a) -> n:int * e:'a -> 'a
```

```
//h computes n! * e

//2.4
//A: seq <int>
//B: seq <int * int>
//C: seq <int * int>

//X: [0; 1; 2; 3]
//Y: [(0, 0); (1, 0); (1, 1); (2, 0); (2, 1); (2, 2)]
//Z: [(0, 0); (1, 0); (0, 1); (2, 0); (1, 1); (0, 2); (3, 0); (2, 1); (1, 2); (0, 3)]


//3.1
type Title = string;;
type Section = Title * Elem list
and Elem = Par of string | Sub of Section;;
type Chapter = Title * Section list;;
type Book = Chapter list;;
let section11 = ("Ba kground", [Par "bla"; Sub(("Why programming", [Par "Bla."]))]);;
let section12 = ("An example", [Par "bla"; Sub(("Spe ial features", [Par "Bla."]))]);;
let section21 = ("Fundamental on epts",[Par "bla"; Sub(("Mathemati al ba kground", [Par "Bla."]))]);;
let section22 = ("Operational semanti s",[Sub(("Basi s", [Par "Bla."])); Sub (("Appli ations", [Par "Bla."]))]);;
let section23 = ("Further reading", [Par "bla"]);;
let section31 = ("Overview", [Par "bla"]);;
let section32 = ("A simple example", [Par "bla"]);;
let section33 = ("An advan ed example", [Par "bla"]);;
let section34 = ("Ba kground", [Par "bla"; Sub(("Why programming", [Par "bla"; Sub(("Why programming", [Par "Bla."]))]))]);;
let section41 = ("Status", [Par "bla"]);;
let section42 = ("What's next?", [Par "bla"]);;
let h1 = ("Introdu tion", [section11;section12]);;
let h2 = ("Basi  Issues", [section21;section22;section23]);;
let h3 = ("Advan ed Issues", [section31;section32;section33;section34]);;
let h4 = ("Con lusion", [section41;section42]);;
let book1 = [ h1; h2; h3; h4];;

//3.1
let rec maxL n = function
    | [] -> n
    | x::xs when x > n -> maxL x xs
    | x::xs -> maxL n xs;;

maxL 0 [1;2;9;7;4;8;9]


//3.2
let overview bk =
    let (t,s) = List.unzip bk
    t;;

overview book1


//3.3
let rec allElements n = function
    | [] -> n
    | (Par p)::es -> allElements n es
    | (Sub (t,(x::xs)))::es -> maxL 0 ([allElements (n+1) xs] @ [allElements (n) es])
    | _ -> failwith "idk";;


let depthSection = function
    | (t, []) -> 2
    | (t, es) -> allElements 2 es;;

let depthElement = function
    | Par p -> 2
    | Sub (t,es) -> allElements 2 es;;


depthElement (Sub(("Why programming", [Par "bla"; Sub(("Why programming", [Par "Bla."]))])))

let rec depthSecs = function
    | [] -> []
    | s::sx -> (depthSection s)::(depthSecs sx);;

let depthChapter = function
    | t, [] -> 1
    | t,sx -> maxL 1 (depthSecs sx);;

depthChapter h1
depthChapter h2
depthChapter h3
depthChapter h4

let rec chapters = function
    | [] -> [1]
    | c::cs -> (depthChapter c)::(chapters cs);;

let depthBook bk = maxL 0 (chapters bk);;

depthBook book1

type Numbering = int list
type Entry = Numbering * Title
type Toc = Entry list

```

```
//let rec makeSubs sq n = function
//    | [] -> []
//    | (Par p)::es -> []
//    | (Sub (t,es'))::es ->  let newsq = Seq.append sq (Seq.singleton n)
//                            let m = (List.ofSeq newsq)
//                            let subs = (m, t)::(makeSubs newsq (n+1) es')
//                            subs::(makeSubs sq (n+1) es);;


let rec makeSections n m = function
    | [] -> []
    | ((t,es)::sx) ->   ([n;m], t)::(makeSections n (m+1) sx);;


let rec makeChapters n = function
    | [] -> []
    | (t,sx)::cs -> let sections = ([n],t)::(makeSections n 1 sx)
                    sections::(makeChapters (n+1) cs);;

let tocB bk = makeChapters 1 bk

tocB book1
```