```
1 //1.1
2 let inv ms =
       (ms = List.distinct ms) &&
       (List.forall (fun (e,n) \rightarrow n > 0) ms);;
 6 //1.2
 7 let rec insert e n = function
        | [] -> []
        (x,y)::xs when x=e \rightarrow (x,y+n)::xs
        | (x,y)::xs -> (x,y)::(insert e n xs);;
10
11
12 //1.3
13
14 let rec numberOf e = function
15
        | [] -> 0
        | (x,y)::xs when x=e \rightarrow y
17
        (x,y)::xs -> numberOf e xs;;
18
19 //1.4
20 let delete e ms = List.filter (fun (x,y) -> e <> x) ms;;
21
22 //1.5
23 let rec union ms1 ms2 =
       match (ms1, ms2) with
25
        | [], _ -> ms2
26
        | _, [] -> ms1
27
        |(x,y)::xs, ms2 \rightarrow let msnew = insert x y ms2
28
                            union xs msnew;;
29
30 //1.6
31 let inv2 ms = Map.forall (fun e n -> n > 0) ms;;
32
33 let insert2 e n ms =
34
       if Map.containsKey e ms then
                                         let n' = Map.find e ms
35
                                         Map.add e (n+n') ms
36
       else Map.add e n ms;;
37
38 let delete2 e ms = Map.remove e ms;;
39
40 //2.1
41 //f int -> 'a list -> (int * 'a) list
42 //Makes a new of tuples (i[j], x[j]) where i[j] = i[j-1] * i[j-1]
43
44 //g ('a -> bool) -> 'a Tree _> 'a Tree option
45 //Finds the first element in the tree which satisfies p
46 //and returns the tree traversed so far
47
48 let rec f i = function
49
        | [] -> []
50
        | x::xs -> (i,x)::f (i*i) xs;;
51
52 type 'a Tree = | Lf
```

```
53
                    | Br of 'a Tree * 'a * 'a Tree;;
 54
 55
 56 f 3 [1;2;3;4;5]
 57
 58 let rec g p = function
 59
         | Lf -> None
         | Br(\_,a,t)  when p a -> Some t
 60
 61
         | Br(t1,a,t2) ->
                             match g p t1 with
 62
                              None -> g p t2
 63
                              res -> res;;
 64
 65 //2.2.1 tail-recursive
 66 let rec fA i a = function
 67
         | [] -> List.rev a
 68
         | x::xs -> fA (i*i) ((i,x)::a) xs;;
 69
 70 fA 3 [] [1;2;3;4;5]
 71
 72 //2.2.1 continuation-based
 73 let rec fC i c = function
 74
         | [] -> c []
 75
         | x::xs \rightarrow fC (i*i) (fun v \rightarrow c((i,x)::v)) xs;;
 76
 77 fC 3 id [1;2;3;4;5]
 78
 79 //2.3
 80 let rec h f (n,e) =
 81
         match n with
 82
         | 0 -> e
 83
         | _ -> h f (n-1, f n e);;
 84
 85 let A = Seq.initInfinite id;;
 86
 87 let B = seq \{ for i in A do \}
 88
                          for j in seq \{0...i\} do
 89
                                  yield (i,j)};;
 90
 91 let C = seq { for i in A do
                          for j in seq \{0...i\} do
 93
                                  yield (i-j,j)};;
 94
 95 let X = Seq.toList (Seq.take 4 A)
 96 let Y = Seq.toList (Seq.take 6 B)
 97 let Z = Seq.toList (Seq.take 10 C)
 98
 99 h (*) (4,2)
100
101 //2.3
102 //h (*) (4,1) = (4*3*2*1)*(1) = 24
103 //h (*) (4,1) = (4*3*2*1)*(2) = 48
104 //h (int -> 'a -> 'a) -> n:int * e:'a -> 'a
```

```
...\Dropbox\05 Semester\Funktionsprogrammering\Exam\2013.fsx
```

```
105 //h computes n! * e
106
107 //2.4
108 //A: seq <int>
109    //B: seq <int * int>
110 //C: seq <int * int>
112 //X: [0; 1; 2; 3]
113 //Y: [(0, 0); (1, 0); (1, 1); (2, 0); (2, 1); (2, 2)]
114 //2: [(0, 0); (1, 0); (0, 1); (2, 0); (1, 1); (0, 2); (3, 0); (2, 1); (1, 2); (0, \nearrow
        3)]
115
116
117 //3.1
118 type Title = string;;
119 type Section = Title * Elem list
120 and Elem = Par of string | Sub of Section;;
121 type Chapter = Title * Section list;;
122 type Book = Chapter list;;
123 let section11 = ("Ba kground", [Par "bla"; Sub(("Why programming", [Par
       "Bla."]))]);;
124 let section12 = ("An example", [Par "bla"; Sub(("Spe ial features", [Par
       "Bla."]))]);;
125 let section21 = ("Fundamental on epts",[Par "bla"; Sub(("Mathemati al ba
       kground", [Par "Bla."]))]);;
126 let section22 = ("Operational semanti s",[Sub(("Basi s", [Par "Bla."])); Sub
       (("Appli ations", [Par "Bla."]))]);;
127 let section23 = ("Further reading", [Par "bla"]);;
128 let section31 = ("Overview", [Par "bla"]);;
129 let section32 = ("A simple example", [Par "bla"]);;
130 let section33 = ("An advan ed example", [Par "bla"]);;
131 let section34 = ("Ba kground", [Par "bla"; Sub(("Why programming", [Par "bla";
       Sub(("Why programming", [Par "Bla."]))]));;
132 let section41 = ("Status", [Par "bla"]);;
133 let section42 = ("What's next?", [Par "bla"]);;
134 let h1 = ("Introdu tion", [section11;section12]);;
135 let h2 = ("Basi Issues", [section21;section22;section23]);;
136 let h3 = ("Advan ed Issues", [section31;section32;section33;section34]);;
137 let h4 = ("Con lusion", [section41;section42]);;
138 let book1 = [ h1; h2; h3; h4];;
139
140 //3.1
141 let rec maxL n = function
142
         | [] -> n
143
         x::xs when x > n -> maxL x xs
144
         x::xs -> maxL n xs;;
145
146 maxL 0 [1;2;9;7;4;8;9]
147
148
149 //3.2
150 let overview bk =
```

```
...\Dropbox\05 Semester\Funktionsprogrammering\Exam\2013.fsx
```

```
4
```

```
let (t,s) = List.unzip bk
151
152
        t;;
153
154 overview book1
155
156
157 //3.3
158 let rec allElements n = function
159
         [] -> n
         | (Par p)::es -> allElements n es
160
161
         | (Sub (t,(x::xs)))::es -> maxL 0 ([allElements (n+1) xs] @ [allElements (n) \Rightarrow
162
         | _ -> failwith "idk";;
163
164
165 let depthSection = function
166
         (t, []) -> 2
         (t, es) -> allElements 2 es;;
167
168
169 let depthElement = function
         | Par p -> 2
170
171
         | Sub (t,es) -> allElements 2 es;;
172
173
    depthElement (Sub(("Why programming", [Par "bla"; Sub(("Why programming", [Par
       "Bla."]))])))
175
176 let rec depthSecs = function
         | [] -> []
178
         | s::sx -> (depthSection s)::(depthSecs sx);;
179
180 let depthChapter = function
181
         | t, [] -> 1
182
         t,sx -> maxL 1 (depthSecs sx);;
183
184 depthChapter h1
185 depthChapter h2
186 depthChapter h3
187 depthChapter h4
189 let rec chapters = function
190
         | [] -> [1]
191
         c::cs -> (depthChapter c)::(chapters cs);;
192
193 let depthBook bk = maxL 0 (chapters bk);;
194
195 depthBook book1
196
197 type Numbering = int list
198 type Entry = Numbering * Title
199 type Toc = Entry list
200
```

```
201 //let rec makeSubs sq n = function
202 //
          | [] -> []
203 //
          | (Par p)::es -> []
204 //
          | (Sub (t,es'))::es -> let newsq = Seq.append sq (Seq.singleton n)
                                  let m = (List.ofSeq newsq)
205 //
206 //
                                  let subs = (m, t)::(makeSubs newsq (n+1) es')
207 //
                                  subs::(makeSubs sq (n+1) es);;
208
209
210 let rec makeSections n m = function
211
         | [] -> []
        | ((t,es)::sx) → ([n;m], t)::(makeSections n (m+1) sx);;
212
213
214
215 let rec makeChapters n = function
216
        | [] -> []
         (t,sx)::cs -> let sections = ([n],t)::(makeSections n 1 sx)
217
218
                        sections::(makeChapters (n+1) cs);;
219
220 let tocB bk = makeChapters 1 bk
221
222 tocB book1
223
224
225
```