```
1  //week4
2  open System
3  // 5.3
4  // foldBack last->first
5  // let p1 x = x > 2
6  let sum1 (p1, xs) =
7      List.foldBack (fun x t -> if p1 x then t + x else t) xs 0
8
9  sum1 ((fun x -> x < 2), [ 5; 5; 2; 6; 1; 0 ])
10
11 // fold first->last
12 // let p2 x = x > 1
13 let sum2 (p2, xs) =
14     List.fold (fun t x -> if p2 x then t + x else t) 0 xs
15
16 sum2 ((fun x -> x > 1), [ 5; 5; 2; 6; 1; 0 ])
17
18 // The function findArticle is replaced by an application of List.tryFind
19 // Cash register
20 // The following declaration names a register
21 let reg =
22     [ ("a1", ("cheese", 25))
23       ("a2", ("herrring", 4))
24       ("a3", ("soft drink", 5)) ]
25 // The following declaration names a purchase:
26 let pur = [ (3, "a2"); (1, "a1") ]
27 (*// findArticle: ArticleCode->Register->ArticleName*Price
28 let rec findArticle ac = function
29     | (ac',adesc)::_ when ac=ac' -> adesc
30     | _::reg -> findArticle ac reg
31     | _ -> failwith(ac + "is an unknown article code")*)
32 // List.tryFind operates the each element in reg, so sgould be fun eachElement  -↵
     > ...
33 let findArticle (ac, reg) =
34     let temp =
35         List.tryFind (fun (ac', adesc) -> ac = ac') reg // T' option
36
37     let (no, info) = temp.Value // .Value get T'
38     info
39
40 findArticle ("a1", reg)
41 // Exception handling 1
42 (*let findArticle(ac,reg) =
43     let temp = List.tryFind (fun (ac',adesc) -> ac=ac') reg // T' option
44     let (no, info) =
45         try
46             temp.Value // .Value get T'
47         with
48         | :? System.NullReferenceException ->  printfn "%s is an unknown article ↵
```

```
            code" ac; ("None",("None",0))
    info
findArticle("a",reg)
// Exception handling 2
exception NullReferenceException of string //'string' stores error description
let findArticle(ac,reg) =
    let temp = List.tryFind (fun (ac',adesc) -> ac=ac') reg // T' option
    match temp with
    | None -> raise (NullReferenceException("unkownArticle"))
    | _ -> let (no, info) = temp.Value
           info// .Value get T'
findArticle("a",reg)
 *)

// The function makeBill is declared using List.foldBack
// makeBill: Register->Purchase->Bill
(*let rec makeBill reg = function
    | [] -> ([],0)
    | (np,ac)::pur -> let (aname,aprice) = findArticle ac reg
                      let tprice = np*aprice
                      let (billt1,sumt1) = makeBill reg pur
                      // billt1 is (np, name, tprice) list, sumt1 is tprice
                      ((np,aname,tprice)::billt1,tprice+sumt1)*)

let makeBill (reg, pur) =
    (*  let (aname,aprice) = findArticle ac reg
    let tprice = List.foldBack (fun (np,ac) t -> t + np.aprice) pur 0
    let (billt1,sumt1) = makeBill reg pur*)
    List.foldBack
        (fun (np, ac) (s, t) ->
            let (aname, aprice) = findArticle (ac, reg)
            let tprice = np * aprice
            ((np, aname, tprice) :: s, t + tprice))
        pur
        ([], 0)

makeBill (reg, pur)

// 2015 winter Problem2
// 1
let f (x) = x-2
let rec mixMap f xs ys =
    match (xs, ys) with
    | (xh :: xt, yh :: yt) -> f xh yh :: mixMap f xt yt
    | ([],[]) -> []
// mixMap f [1;3;9;0] [5;4;3;1]
// 2
let g y = y + 5
let unmixMap f g xys = List.foldBack (fun (x,y) (xs,ys) -> (f x :: xs, g y ::    ↵
```

```
     ys)) xys ([],[])
97  unmixMap f g [(1,2);(3,5);(1,9);(0,4)]
98
99
```