```fsharp
// week 8&9
// 2019 summer P1 Q6 high-order function
let f x yl = List.map (fun y -> (x, y)) yl
f "a" [ 1; 2; 3 ]
//
type exp =
    | C of int
    | BinOp of exp * string * exp
    | Id of string
    | Def of string * exp * exp

(*
[1;2;3]
[(C 1,"w",C 7);(C 0,"n",C 7);(C 6,"o",C 7)]
*)

// 2015 Dec P1 1.2
type Appliance = string
type Usage = Appliance * int
let ad1 = ("washing machine", 1)
let ad2 = ("coffee machine", 1)
let ad3 = ("dishwasher", 2)
let ats = [ ad1; ad2; ad3; ad1; ad2 ]

let rec durationOf a ats =
    match (a, ats) with
    | ((_, t), []) -> 0
    | ((_, t), atsh :: atst) when a = atsh -> t + (durationOf a atst)
    | ((_, t), atsh :: atst) -> durationOf a atst

durationOf ad2 ats

// 1.1
let rec inv ats =
    match ats with
    | [] -> true
    | (_, t) :: atst when t > 0 -> inv atst
    | _ -> false

inv ats

// 1.3
let rec wellFormed ats =
    List.forall (fun a -> (durationOf a ats) <= 24) ats
    && inv ats

wellFormed ats

// week 8
```

```
50 // 2011 P2
51 // 2. toString: exp -> string
52 let rec toString =
53     function
54     | C n -> string n
55     | BinOp (e1, s, e2) -> "(" + (toString e1) + s + (toString e2) + ")"
56
57 toString (BinOp(C 3, "+", BinOp(C 5, "*", C 2)))
58 toString (BinOp(BinOp(C 2, "*", C 6), "+", BinOp(BinOp(C 9, "+", C 4), "*", C
     2)))
59
60 // 3. Extracting the set of operators from an expression
61 let rec ops =
62     function
63     | C n -> []
64     | BinOp (e1, s, e2) -> [ s ] @ (ops e1) @ (ops e2)
65
66 ops (BinOp(BinOp(C 2, "*", C 6), "+", BinOp(BinOp(C 9, "+", C 4), "*", C 2)))
67
68 // 4. isDef: exp -> bool
69 let rec f id e =
70     match e with
71     | BinOp (e1, s, e2) -> isDef id e1 && isDef id e2
72     | Def (id, e1, e2) -> isDef id e1 && isDef id e2
73     | _ -> isDef id e
74
75 and isDef id e =
76     match e with
77     | C n -> true
78     | Id x when id = x -> true
79     | _ -> f id e
80
81 isDef "" (Def("x", C 5, BinOp(Id "y", "+", Id "x")))
82 isDef "" (Def("y", C 5, BinOp(Id "y", "+", Id "x")))
83 isDef "" (Def("y", C 5, BinOp(Id "y", "+", Id "y")))
84
85 // 2015 Dec P3
86 type Name = string
87 type Flow = int // can be assumed positive in below questions
88
89 type River = R of Name * Flow * Tributaries
90 and Tributaries = River list
91 // 1. Declare F# values
92 let riv =
93     R(
94         "R",
95         10,
96         [ R("R1", 5, [])
97           R("R2", 15, [ R("R4", 2, []) ])
```

```
 98            R("R3", 8, []) ]
 99     )
100
101 let riv3 = R("R3", 8, [])
102 let riv2 = R("R2", 15, [ R("R4", 2, []) ])
103 // 2. cotains: Name->River->bool
104 let rec contains n r =
105     match r with
106     | R (n0, _, _) when n = n0 -> true
107     | R (_, _, t) -> false || (trib n t)
108
109 and trib n r =
110     match r with
111     | [] -> false
112     | rh :: rt -> (contains n rh) || (trib n rt)
113
114 contains "R" riv3
115
116 // 3. allNames: River -> Name list
117 let rec allNames r =
118     match r with
119     | R (n, _, t) -> n :: (tribNames t)
120
121 and tribNames r =
122     match r with
123     | [] -> []
124     | rh :: rt -> (allNames rh) @ (tribNames rt)
125
126 allNames riv2
127
128
129 // 4. totalFlow: River -> Flow
130 let rec totalFlow r =
131     match r with
132     | R (_, f, t) -> f + (tribFlow t)
133
134 and tribFlow r =
135     match r with
136     | [] -> 0
137     | rh :: rt -> (totalFlow rh) + (tribFlow rt)
138
139 totalFlow riv
140
141 // 5. mainSource: River -> (Name*Flow)
142 let mainSource r =
143     let ms = ("n", 0)
144
145     let rec source r ms =
146         match (r, ms) with
```

```
147         | (R (n, f, t), (_, mf)) when f > mf -> tribSource t (n, f)
148         | (R (_, _, t), ms) -> tribSource t ms
149
150     and tribSource r ms =
151         match (r, ms) with
152         | ([], ms) -> ms
153         | (rh :: rt, ms) -> tribSource rt (source rh ms)
154
155     source r ms
156
157 mainSource riv
158
159 // 6. tryInsert: Name -> River -> River -> River option
160 let tryInsert n t r =
161     if not (contains n r) then
162         None
163     else
164         let rec insert n t r =
165             match r with
166             | R (n0, f, t0) when n = n0 -> R(n0, f, t :: t0)
167             | R (n0, f, t0) -> R(n0, f, (tribInsert n t t0))
168
169         and tribInsert n t r =
170             match r with
171             | [] -> []
172             | th :: tt -> (insert n t th) :: (tribInsert n t tt)
173
174         Some(insert n t r)
175
176 tryInsert "R" (R("R5", 100, [])) riv
177
```