TUTORIAL 3 PEMROGRAMAN DEKLARATIF

Tipe data Tree yang akan digunakan adalah sebagai berikut

```
type tree<'a> =
    |Lf
    |Br of 'a * tree<'a> * tree<'a>;;
```

1. Buatlah sebuah fungsi “leafMenjadiKucing” yang akan mengganti seluruh isi leaf dengan branch yang berisi “kucing” dan kedua anak kiri dan kanannya adalah leaf. Tipe data dari fungsi tersebut adalah

tree<string> -> tree<string>

Contoh:

leafMenjadiKucing(Lf) -> Br(“kucing”,Lf,Lf)

leafMenjadiKucing(Br(“sambel”,Lf,Lf)) ->

Br(“sambel”,Br(“kucing”,Lf,Lf),Br(“kucing”,Lf,Lf))

```
let rec leafMenjadiKucing = function
 |Lf -> Br("kucing",Lf,Lf)
 |Br(z,kiri,kanan) ->
Br(z,leafMenjadiKucing(kiri),leafMenjadiKucing(kan
an));;
```

2. Buatlah sebuah fungsi “listOfLeaves” yang akan mengambil seluruh isi elemen branch yang memiliki anak kanan dan kiri leaf. Urutan tidak masalah. Tipe Data dari fungsi tersebut adalah

tree<'a> -> 'a list

Contoh:

listOfLeaves(Br(1,Br(2,Lf,Lf),Br(3,Lf,Br(4,Lf,Lf)) -> [2;4]

```
let rec listOfLeaves = function
 |Lf -> []
 |Br(z,Lf,Lf) -> [z]
 |Br(z,x,y)->listOfLeaves(x)@listOfLeaves(y);;
```

3. Buatlah sebuah fungsi “bagiTree” yang akan membagi seluruh isi elemen dari sebuah binary tree dengan sebuah bilangan. Tipe data dari fungsi tersebut adalah:

tree<int>*int -> tree<int>

Contoh:

bagiTree(Br(10,Br(5,Lf,Lf),Br(15,Lf,Lf)),5) = Br(2,Br(1,Lf,Lf),Br(3,Lf,Lf))

```
let rec bagiTree = function
 |(Lf,x) -> Lf
 |(Br(z,kiri,kanan),x) ->
Br(z/x,bagiTree(kiri,x),bagiTree(kanan,x))
```

4. Buatlah Sebuah Fungsi “filterHeight” yang akan mengembalikan tree sampai kedalaman tertentu dan sisanya dibuang. Tipe data dari fungsi tersebut adalah

tree<’a> * int -> tree<’a>

Contoh:

filterHeight(Br(1,Br(2,Lf,Lf),Br(3,Lf,Lf)),1) = Br(1,Lf,Lf)

```
let rec filterHeight = function
 |(Lf,x) -> Lf
 |(_,0) -> Lf
 |(Br(z,kiri,kanan),x) ->
Br(z,filterHeight(kiri,x-1),filterHeight(kanan,x-
1)
```

5. Buatlah sebuah fungsi “isBST” yang akan mengembalikan apakah tree tersebut merupakan sebuah binary search tree apa bukan. Tipe data dari fungsi tersebut adalah:

tree<int> -> bool

Contoh:

isBST(Br(5,Lf,Lf))= true

isBST(Br(5,Br(3,Lf,Lf),Br(4,Lf,Lf))= false

```
let rec isBST = function
 |Lf -> true
 |Br(z,Lf,Lf) -> true
 |Br(z,Br(x,a,s),Lf)->x<=z && isBST(Br(x,a,s))
 |Br(z,Lf,Br(x,a,s))->z<x && isBST(Br(x,a,s))
 |Br(z,Br(x,a,s),Br(y,q,w)) -> x<=z&& z<y &&
isBST(Br(x,a,s))&& isBST(Br(y,q,w))
```

6.Buatlah sebuah fungsi yang akan mencari elemen terbesar dari BST “maxBST”

tipe data dari fungsi tersebut:

tree<int> -> int

Contoh:

maxBST(Br(2,Br(1,Lf,Lf),Br(3,Lf,Lf))) = 3

```
let rec isBST = function
 |Lf -> -1
 |Br(z,_ Lf)-> z
 |Br(z,_,x))->maxBST(x)
```

7. Tentukan tipe dari fungsi di bawah ini:

```
let rec sumt = function
    | Lf -> 0
    | Br (x, lt, rt) -> x + (sum lt) + (sum rt)
```

```
tree<int> -> int
```

8. Tentukan tipe dari fungsi di bawah ini:

```
let rec zipt = function
     | (Lf, Lf) -> Lf
    | (Lf, Br(_, _, _)) -> Lf
    | (Br(_, _, _), Lf) -> Lf
    | (Br (x, lta, rta), Br (y, ltb, rtb)) -> Br ((x, y), zipt
(lta, ltb), zipt(rta, rtb))
```

```
(tree<'a>, tree<'b>) -> tree<('a, 'b)>
```

9. Lakukanlah trace pada fungsi zipt diatas dengan pemanggilan

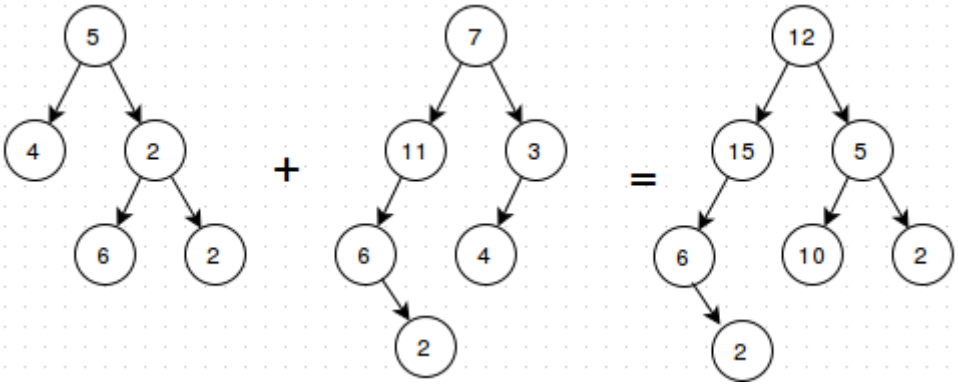
```
zipt (Br(44,
    Br (3, Lf, Lf),
    Br (2,
      Br (10, Lf, Lf),
      Br (2, Lf, Lf))),
Br(12,
    Br(5,
      Br(2, Lf, Lf),
      Lf),
    Br(6,
      Lf,
                          Br(2, Lf, Lf))))
```

```
Jawaban akhir:
Br((44, 12),
    Br((3,5), Lf, Lf),
    Br((2,6),
      Lf,
      Br((2,2), Lf, Lf)))
```

10. Buatlah fungsi untuk mengembalikan list yang berisi elemen-elemen BST secara terurut menaik!

```
let rec inorder = function
    | Lf -> []
    | Br (x, lt, rt) -> (inorder lt) @ x::(inorder
rt)
```

11. Buatlah fungsi untuk menjumlahkan dua buah tree! Contoh:



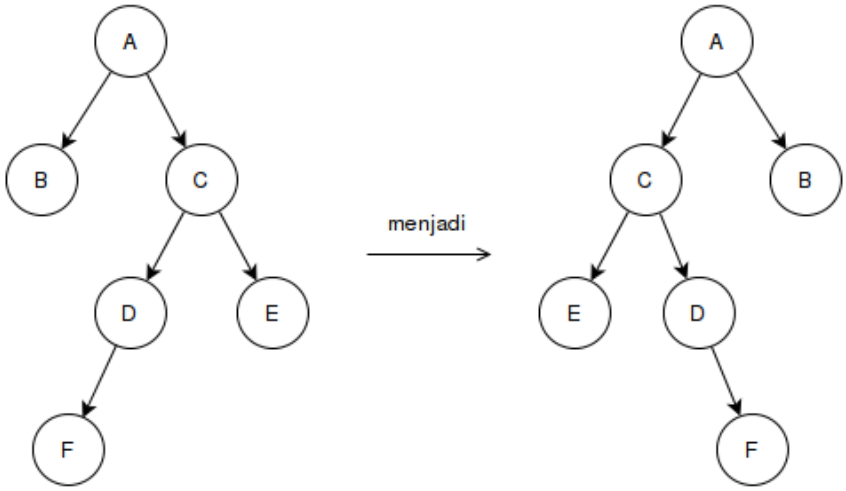

```
let rec sumtree = function

    | (Lf, Lf) -> Lf
    | (Lf, Br(y, ltb, rtb)) -> Br (y, ltb, rtb)
    | (Br(x, lta, rta), Lf) -> Br(x, lta, rta)
    | (Br(x, lta, rta), Br(y, ltb, rtb)) ->
Br(x+y, sumtree (lta, rta), sumtree(ltb, rtb))
```

12. Buatlah fungsi untuk mengembalikan list yang berisi seluruh elemen pada suatu level dalam tree! Urutan elemen tidak penting.

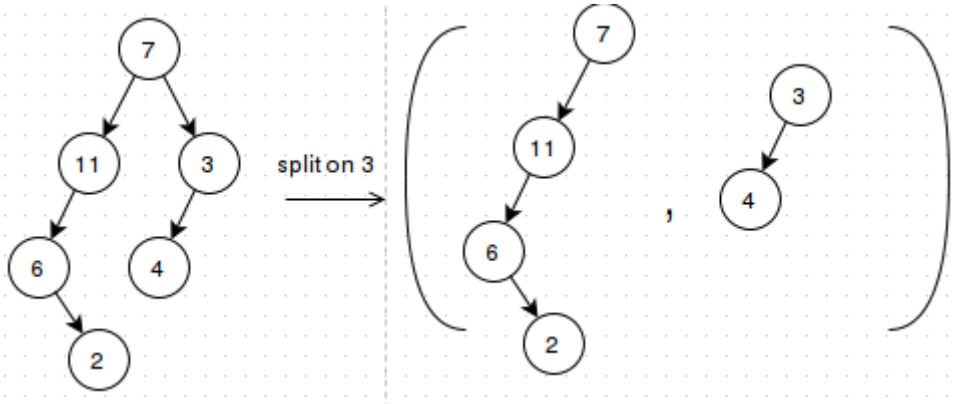
```
let rec getNodesOnLevel = function
    | (Lf, n) -> []
    | (Br (x, _, _), 0) -> [x]
    | (Br (x, lt, rt), n) -> x::(getNodesOnLevel
(lt, n-1)) @ getNodesOnLevel (rt, n-1)
```

13. Buatlah fungsi untuk membalikkan (me-mirror) tree. Contoh:




```
let rec mirror = function
    | Lf -> Lf
    | Br (x, lt, rt) -> Br (x, mirror rt, mirror
lt)
```

14.Buatlah fungsi untuk memotong tree di suatu elemen yang dicari. Kembaliannya adalah tuple tree sisa dan subtree hasil potongan. Setiap elemen dalam tree dijamin unik. Contoh:

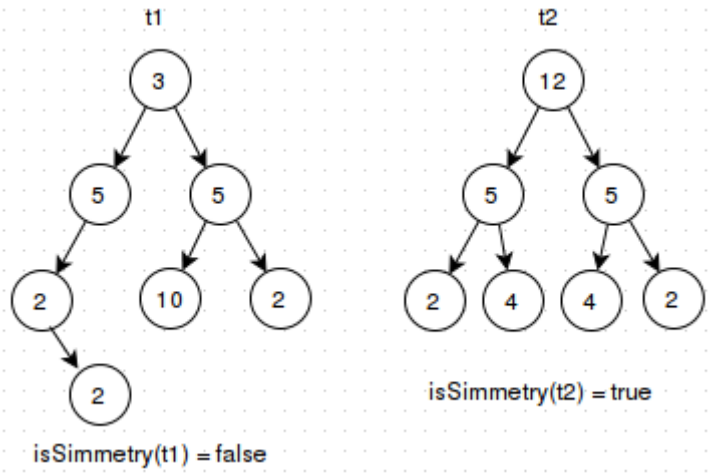


Method signature: (tree<int>, int) -> (tree<int>, tree<int>)

```
let rec splitOn = function
    | (Lf, k) -> (Lf, Lf)
    | (Br (x, lt, rt), k) ->
```

```
    if x = k then
        (Lf, Br(x, lt, rt))
    else
        let (lori, lres) = splitOn(lt, x)
            (rori, rres) = splitOn(rt, x)
        if lres = Lf then
            (Br (x, lori, rori), rres)
        else
            (Br (x, lori, rori), lres)
```

15. Buatlah fungsi untuk mengecek apakah suatu binary tree adalah simetris. Contoh:




```
let rec isSimmetry t =
    let rec isMirror = function
      | (Lf, Lf) -> true
      | (Lf, Br(_, _, _)) -> false
      | (Br(_, _, _), Lf) -> false
      | (Br(x, lta, rta), Br(y, ltb, rtb)) -> x =
y && isMirror (lta, rtb) && isMirror (rta, ltb)
    isMirror (t, t)
```

16.Buatlah fungsi yang menerima sebuah tree dan mengembalikan tree yang nilai di tiap node nya merupakan penjumlahan seluruh elemen di subtree node tersebut

Contoh:

.

```
Sumsub (Br (5, Br (4, Lf, Lf), Br (2, Br (6, Lf, Lf), Br (2, Lf, Lf)))) = Br
(19, Br (4, Lf, Lf), Br (10, Br (6, Lf, Lf), Br (2, Lf, Lf)))
```

```
let rec sumsub = function
 | Lf -> Lf
 | Br (x, Lf, Lf) -> Br(x, Lf, Lf)
 | Br (x, lt, rt) ->
    let slt = sumsub lt
    let srt = sumsub rt
    let (Br (suml, lta, rta)) = slt
    let (Br (sumr, ltb, rtb)) = rlt
    Br (x + suml + sumr, slt, srt);;
```

17. Buatlah sebuah fungsi “irisanTree” yang akan mengembalikan sebuah list yang berisi elemen-elemen yang sama pada dua buah binary search Tree.

tree<int>*tree<int> -> int list

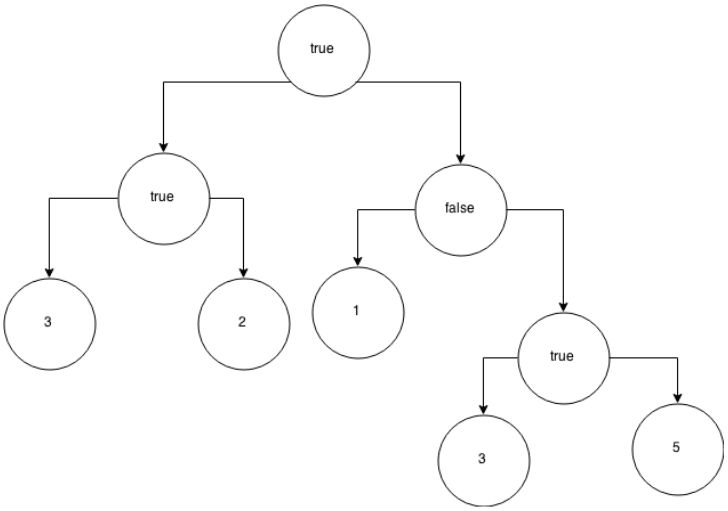
Contoh:

irisanTree(Br(2,Br(1,Lf,Lf),Lf),Br(2,Lf,Br(3,Lf,Lf)))=[2]

```
Let rec irisanList = function
([],_)->[]
(_,[])->[]
(x::xs,y::ys)-> if x=y then x::irisanList(xs,ys) else if x<y then
irisanList(xs,y::ys) else irisanList(x::xs,ys)

let irisanTree (a,b) = irisanList(inorder(a),inorder(b))
```

18. Buatlah sebuah datatype baru “mathTree” yang akan menerima data type sebagai berikut




```
type mathTree=
   |Lf of int
   |Br of bool * mathTree *mathTree
```

19. Dengan datatype baru pada nomor 18. Lakukan operasi matematka pada setiap branch untuk contoh pada nomor 18 maka. (3+2)+(1-(3+5)) false menandakan pengurangan true menandakan penambahan

mathTree -> int

```
Let rec operasi = function
|Lf x -> x
|Br (true,x,y) -> operasi x + operasi y
|Br (false,x,y) -> operasi x - operasi y
```

20. Lakukan tracing untuk contoh mathtree pada nomor 18 dengan fungsi yang anda buat pada nomor 19!