Latihan – Programming with Lists

Petunjuk: Dalam mengerjakan permasalahan berikut, Anda mungkin bisa saja menggunakan ***List Comprehension***. Namun, Cobalah belajar untuk mendefinisikannya menggunakan konsep ***recursion*** (primitive or general recursion). Anda boleh menggunakan ***local definitions*** jika dibutuhkan.

Generating List

1. Definisikan fungsi **`evenN :: Int -> [Int]`** sehingga **`evenN n`** akan menghasilkan list yang berisi "n bilangan genap pertama yang positif" (n >= 0).

Contoh : **`evenN 5`** akan menghasilkan **`[2,4,6,8,10]`**

2. Definisikan fungsi **`doubleList :: [Int] -> [int]`** sehingga

```
doubleList [x1, x2, x3, ..., xn] = [2*x1, 2*x2, 2*x3, ..., 2*xn]
```

Contoh : **`doubleList [1,2,3,4]`** akan mengembalikan **`[2,4,6,8]`**

Properti List

3. Definisikan fungsi **`numOdd :: [Int] -> Int`** yang mengembalikan nilai banyaknya bilangan ganjil dalam sebuah list of integers.

Contoh : **`numOdd [2,3,4,5,6,7,8]`** akan mengembalikan 3

4. Definisikan fungsi **`multiplicity x ls`** yang bertipe **`Int -> [Int] -> Int`**. Fungsi ini mengembalikan nilai banyaknya kemunculan elemen **x** di dalam list **ls**.

Contoh : **`multiplicity 2 [2,3,4,2,5,2,6,2]`** akan mengembalikan 4

5. Definisikan fungsi **`listParity ls :: [a] -> Bool`** yang mengembalikan **True** jika panjang list adalah genap dan mengembalikan **False** jika sebaliknya. **Hint**: agar lebih elegan, coba gunakan operasi **not** dalam hal ini.

Contoh : **`listParity [3,2,4,5]`** akan mengembalikan **True**

Removing and Filtering

6. Definisikan fungsi **`removeEven :: [Int] -> [Int]`** yang membuang semua elemen genap pada sebuah list.

Contoh : **`removeEven [1,2,3,4,5,6]`** akan mengembalikan **`[1,3,5]`**

7. Deklarasikan fungsi **`throwLastElmt : [a] -> [a]`** yang membuang **elemen terakhir** pada list. Perhatikan bahwa list masukan harus **minimal** punya 1 elemen.

Contoh : **`throwLastElmt [1,2,3,4,5]`** akan mengembalikan **`[1,2,3,4]`**

8. Deklarasikan fungsi **`getLastElmt : [a] -> a`** yang mengambil elemen terakhir dari sebuah list. Perhatikan bahwa list masukan harus **minimal** punya 1 elemen.

Contoh : **`getLastElmt [1,2,3,4,5]`** akan mengembalikan **5**

9. Deklarasikan fungsi **`removeFirst x ls`** yang mempunyai tipe **`Int -> [Int] -> [Int]`**. Fungsi akan menghapus **kemunculan pertama x** dalam list **ls**.

Contoh : **`removeFirst 2 [1,2,3,2,4,2,5,2]`** akan mengembalikan **`[1,3,2,4,2,5,2]`**

Inserting Element

10. Deklarasikan fungsi **`addLast x ls`** yang bertipe **`Int -> [Int] -> [Int]`**. Fungsi ini akan menambahkan elemen **x** pada posisi terakhir di list **ls**, jika elemen **x** belum ada pada list. Jika **x** sudah ada pada list, maka **x** tidak akan ditambahkan ke dalam list.

Contoh : **`addLast 4 [1,2,3]`** akan mengembalikan **`[1,2,3,4]`**

`addLast 4 [1,4,2,3]` akan mengembalikan **`[1,4,2,3]`**

11. Deklarasikan fungsi **`insert x ls`** yang bertipe **`Int -> [Int] -> [Int]`**. Kita asumsikan **list ls sudah terurut menaik**. Fungsi insert akan menyisimpkan elemen **x** ke dalam list dengan tetap menjaga keterurutan dari list tersebut.

Contoh : **`insert 5 [1,2,3,7,8,9]`** akan mengembalikan **`[1,2,3,5,7,8,9]`**

Soal Tambahan (Intermediate)

12. Deklarasikan fungsi **`isTerurutMenaik :: [Int] -> Bool`** yang mengembalikan nilai **True** jika list **terurut menaik** dan **False** jika sebaliknya. * terurut menaik semu: untuk input **[2,2,2]** akan mengebalikan **True**.

Contoh : **`isTerurutMenaik [2,4,5,4,6,7,8]`** mengembalikan False.

13. Deklarasikan fungsi **`findMax :: [Int] -> Int`** yang mengembalikan nilai paling **maksimal** dari sebuah list. List diasumsikan mempunyai minimal 1 elemen. Anda bisa menggunakan *local definitions* disini, untuk mendefinisikan fungsi **max2**.

Contoh : **`findMax [1,2,3,4,5,4,3,2,1]`** akan mengembalikan 5

14. Deklarasikan fungsi **`split :: [Int] -> ([Int], [Int])`** sehingga

`split [x1,x2,x3,x4,x5, ...,xn] = ([x1,x3,x5, ...], [x2,x4,x6, ...])`

Contoh : **`split [3,2,4,3,6,7,8]`** akan mengembalikan **`([3,4,6,8], [2,3,7])`**

15. Misalkan, ada sebuah struktur data *modified list* yang mempunyai format berikut

$[(val_1, count_1), (val_2, count_2), (val_3, count_3), ..., (val_n, count_n)]$

Properti dari struktur data tersebut adalah:

- Untuk setiap pair pada list, **`val`** bersifat unik
- **`count`** merupakan informasi ada berapa banyak **val**

Deklarasikan fungsi **`insert' :: Int -> [(Int, Int)] -> [(Int, Int)]`** yang melakukan penyisipan **val** ke dalam list. Jika **val** sudah ada, maka **count** dari **val** tersebut akan dinaikkan **+1**. Jika **val** belum ada pada list, maka akan ditambahkan **val** tersebut ke dalam list dengan **count** awal = **1**.

Contoh :

`insert' 4 [(3,3), (4,2), (2,9)]) = [(3,3), (4,3), (2,9)]`

`insert' 6 [(3,3), (4,2), (2,9)]) = [(3,3), (4,2), (2,9), (6,1)]`