Responsi

Praktikum 2 - Shift 1

made with love by labpro

Soal 1 (Konkat)

Nama File: ListOfCharacter.hs

Header: module ListOfCharacter where

Salinlah definisi list of character dalam file ListOfCharacter.hs.

Buatlah fungsi **konkat** yang menerima masukan 2 buah list of character, misalnya lc1 dan lc2, yang masing-masing mungkin kosong, dan menghasilkan list baru yang merupakan penggabungan lc1 dengan lc2 (lc1 di awal).

Contoh aplikasi dan hasil:

Aplikasi	Hasil
konkat ['s','a','y'] ['a']	"saya"
konkat [] ['a']	"a"
konkat ['a'] []	"a"
konkat [] []	""

Soal 1 (Konkat)

```
module ListOfCharacter where

-- Konkat
-- DEFINISI DAN SPESIFIKASI
-- fungsi yang menerima masukan 2 buah list of character, misalnya lc1 dan lc2, yang masing-masing
mungkin kosong, dan menghasilkan list baru yang merupakan penggabungan lc1 dengan lc2

konkat :: [Char] -> [Char] -> [Char]

-- Realisasi
konkat li1 li2
      | isEmpty li1 = li2
      | otherwise = konkat (init li1) (konso (last li1) li2)

-- Aplikasi
-- konkat ['s','a','y'] ['a']  => "saya"
-- konkat [] ['a'] => "a"
-- konkat ['a'] [] => "a"
-- konkat [] [] => ""
```

Soal 2 (ListOfInteger)

Nama File: ListOfInteger.hs

Header: module ListOfInteger where

Salinlah definisi list of integer dalam file ListOfInteger.hs.

Buatlah fungsi **isMember** yang menerima masukan sebuah list of integer, misalnya l, dan sebuah integer, misalnya x, dan menghasilkan true jika x adalah salah satu member (anggota) dalam list l. Menghasilkan false jika tidak, atau jika list l kosong.

Contoh aplikasi dan hasil:

Aplikasi	Hasil
isMember [] 5	False
isMember [1,2] 5	False
isMember [1,2,3] 1	True

Soal 2 (ListOfInteger)

```haskell
module ListOfCharacter where

--  DEFINISI DAN SPESIFIKASI
{-
  Menerima masukan sebuah list of integer, misalnya l, dan sebuah integer, misalnya x, dan
menghasilkan true jika x adalah salah satu member (anggota) dalam list l. Menghasilkan false jika
tidak, atau jika list l kosong.
-}
isMember :: [Int] -> Int -> Bool

-- REALISASI
isMember l x
  | isEmpty l = False -- Basis: list kosong menghasilkan nilai False berdasarkan spesifikasi
  | head l == x = True -- Basis: Jika elemen pertama list adalah x maka menghasilkan true
  | otherwise = isMember (tail l) x -- Rekuerens, jika tidak, cek elemen sisa (tail)
```

Soal 3 (Kemunculan Nilai Minimum List)

Nama File: ListOfInteger

Header: module ListOfInteger where

Salinlah definisi list of integer dalam file ListOfInteger.hs.

Buatlah realisasi fungsi **jmlMin** yaitu fungsi yang menerima sebuah list of integer tidak kosong, dan menghasilkan pasangan bilangan integer (a,b), dengan a berisi nilai minimum dari seluruh elemen list dan b berisi jumlah kemunculan nilai minimum tersebut pada l. Fungsi ini akan memanggil fungsi **minList** yang menghasilkan nilai minimum dari seluruh elemen list, dan fungsi **nbX** yang menghasilkan banyaknya kemunculan sebuah elemen pada list.

```
-- Definisi dan Spesifikasi

minList :: [Int] -> Int

{- minList l mengembalikan nilai minimum dari seluruh elemen list -}


nbX :: Int -> [Int] -> Int

{- nbX x l menghasilkan banyaknya kemunculan x pada l -}


jmlMin :: [Int] -> (Int,Int)

{- jmlMin l menghasilkan tuple (a,b) dengan:
      a adalah nilai minimum dari elemen-elemen l dan

      b adalah jumlah kemunculan a pada l -}
```

Soal 3 (Kemunculan Nilai Minimum List)

```
module ListOfInteger where
-- DEFINISI DAN SPESIFIKASI
minList :: [Int] -> Int
{- minList l mengembalikan nilai minimum dari seluruh elemen list -}
nbX :: Int -> [Int] -> Int
{- nbX x l menghasilkan banyaknya kemunculan x pada l -}
jmlMin :: [Int] -> (Int,Int)
{- jmlMin l menghasilkan tuple (a,b) dengan:
      a adalah nilai minimum dari elemen-elemen l dan
      b adalah jumlah kemunculan a pada l -}

{ Soal ini sebenarnya hanya looping sederhana. Tapi, gimana cara kita melakukan loop di Haskell?
Yak, benar! Kita harus menggunakan rekursi. Berikut adalah cara untuk menyelesaikan problem rekursi:
 1.   Tentukan base case, kondisi dimana fungsi berhenti / jawaban tersimpel fungsi. Base case
      umumnya elemen 0 atau elemen 1
 2.   Tentukan rekurens, cara mendapatkan solusi dengan menggunakan subproblem fungsi yang sama
 3.   Gabungkan dan profit! }
```

Soal 3 (Kemunculan Nilai Minimum List)

```
-- REALISASI
minList l
   | isOneElmt l = head l
   | (head l) < minList (tail l) = head l
   | otherwise = minList (tail l)
```

```
nbX x l
   | isEmpty l = 0
   | head l == x = 1 + nbX x (tail l)
   | otherwise = nbX x (tail l)
```

```
jmlMin l = ((minList l),
   (nbX (minList l) l))
{ Ini menggabungkan saja, tidak ada rekursi }
```

Penjelasan Algoritma

Base case, kalau **hanya ada 1 elemen**, maka elemen itu adalah **bilangan terkecil** pada list tersebut.

Rekurens, jika ada lebih dari 1, bandingkan **elemen di ujung (head)** dengan elemen terkecil pada **list tanpa elemen head (tail).**

Base case, kalau **tidak ada elemen,** jumlah kemunculan x adalah 0 kali.

Rekurens, selainnya, lihat **apakah elemen head sama dengan x**, *jika iya*, berarti kita sudah menemukan 1 kemunculan dan dilanjutkan dengan pencarian di **list tanpa elemen head (tail),** *jika tidak*, berarti cek di list tanpa elemen head (tail) saja.

Soal 4 (Duel)

Nama header : module Duel where
Nama file : Duel.hs

Tuan Vin adalah seorang koboi. Saat berduel dengan lawannya, ia harus bersiap-siap mendengar kata "desperado". Apabila ia mendengar kata "desperado", maka ia harus mengatakan "BANG".
Bantulah Tuan Vin untuk mengubah kata "desperado" menjadi "BANG" dari sebuah list agar Tuan Vin menang dengan nama fungsi **duel** yang menerima list bertipe String dan mengeluarkan list bertipe String

Contoh :
li = ["one", "two", "desperado", "cowboy", "guns", "horse", "desperado", "desperado", "desperado"]
Hasil Keluaran = ["one", "two", "BANG", "cowboy", "guns", "horse", "BANG", "BANG", "BANG"

Hint: Gunakan operator : untuk melakukan konkatenasi String dengan List of String

Soal 4 (Duel)

```haskell
module Duel where

-- DUEL
-- DEFINISI DAN REALISASI
duel :: [String] -> [String]
{- fungsi yang mengubah kaya "desperado" menjadi "BANG" dari sebuah list
fungsi duel menerima list bertipe string dan mengeluarkan list bertipe list -}
duel li
    | null li = []
    -- apabila list null, mengembalikan list kosong
    | head li == "desperado" = ["BANG"] ++ duel(tail li)
    {- apabila elemen pertama list adalah "desperado" maka akan diganti menjadi "BANG"
    dan dikontatenasi dengan hasil duel sisa elemen yang lain -}
    | otherwise = [head li] ++ (duel(tail li))
    {- apabila elemen pertama list bukan merupakan "desperado" makan akan mengembalikan
    elemen head yang dikontatenasi dengan hasil duel sisa elemen yang lain -}
-- soal di atas menggunakan konkatenasi string dengan list of string serta rekursi
```

Soal 5 (Pajak Makan)

Nama File: ListOfCharacter.hs
Header: module ListOfCharacter where

Tuan Vin ingin makan di sebuah restoran baru, namun ia uangnya terbatas. Harga makanan yang ditampilkan pada menu restoran belum termasuk pajak dan service.

Bantulah Tuan Vin menghitung makanan mana yang bisa ia beli bila diketahui uangnya 200 rupiah, dan pajak makanan adalah 10%.

Buatlah sebuah fungsi **pajakMakan** yang menerima list nama makanan bertipe char dan list harganya bertipe integer.

Fungsi mengeluarkan list char berisi makanan mana saja yang bisa dibeli Tuan Vin setelah kena pajak.

Keterangan:
- Jumlah menu makanan dan daftar harga pasti sama
- Nama menu makanan pasti unik
- Bila tidak ada makanan yang dapat dibeli, fungsi mengembalikan list kosong

Contoh:

```
> pajakMakan ['A','Z','B'] [200,150,100]
"ZB"

>pajakMakan [] []
""
```

Haskell

ListOfCharacter.hs

Soal 5 (Pajak Makan)

```
module ListOfCharacter where
-- DEFINISI DAN SPESIFIKASI
pajakMakan :: [Char] -> [Int] -> [Char]
{- pajakMakan menerima list of Char yang mewakili kode nama makanan dan list of Int yang mewakili
harga makanan. Fungsi ini mengembalikan daftar makanan yang bisa dibeli Tuan Vin (termasuk pajaknya)
dengan uang 200 rupiah -}

{- Ide dari soal ini adalah bagaimana kita melakukan traversal setiap elemen pada list dan mengecek
apakah harga dari makanan tersebut + 10% <= 200. Jika ya, maka kita akan menambahkannya ke dalam
solusi dan lanjut memeriksa elemen lainnya. Berikut adalah step by step penyelesaiannya:
 1.   Tentukan base case, yaitu juga list makanan sudah kosong, maka pajakMakan mengembalikan []
 2.   Untuk kasus jika harga makanan + pajak <= 200, maka makanan ditambahkan ke solusi bersama
      dengan solusi lainnya
 3.   Jika tidak, maka kita hanya perlu mengembalikan solusi dari elemen-elemen setelahnya -}
```

Soal 5 (Pajak Makan)

```
-- REALISASI
pajakMakan :: [Char] -> [Int] -> [Char]
pajakMakan m h
   | isEmpty m = [] {- BASE CASE -}
   | (fromIntegral(head h * 110) / 100) <= 200 = konso (head m) (pajakMakan (tail m) (tail h))
   | otherwise = pajakMakan (tail m) (tail h)
```

Soal 6 (Pecah List Ganjil Genap)

Nama File : ListOfInteger.hs

Header : *module ListOfInteger where*

Salinlah definisi list of integer dalam file ListOfInteger.hs.

Buatlah sebuah fungsi **pecahListGanjilGenap** (definisi, spesifikasi, dan realisasi) yang menerima masukan sebuah list of integer (l) dan mengembalikan tiga buah list of integer (l1, l2, l3). l1 memuat semua elemen l yang merupakan bilangan negatif dan 0, l2 memuat semua elemen l yang merupakan bilangan positif dan ganjil, sedangkan l3 memuat semua elemen l yang merupakan bilangan positif dan genap. Urutan kemunculan elemen pada l1 dan l2 tetap sama dengan urutan elemen pada l.

No	Input	Output
1.	pecahListGanjilGenap [1,3,0,4,-1,4,-9]	([0,-1,-9],[1,3],[4,4])
2.	pecahListGanjilGenap [2,3,4,5]	([],[3,5],[2,4])
3.	pecahListGanjilGenap []	([],[],[])

Soal 6 (Pecah List Ganjil Genap)

```
module ListOfInteger where

-- DEFINISI DAN SPESIFIKASI
pecahListGanjilGenap :: [Int] -> ([Int], [Int], [Int])
{-
Memecah list menjadi (l1,l2,l3).
 1.   l1 memuat semua elemen l yang merupakan bilangan negatif dan 0
 2.   l2 memuat semua elemen l yang merupakan bilangan positif dan ganjil
 3.   l3 memuat semua elemen l yang merupakan bilangan positif dan genap
-}
```

Soal 6 (Pecah List Ganjil Genap)

```
-- REALISASI (MENGGUNAKAN APPROACH PROSEDURAL)
pecahListGanjilGenap l =
   if isEmpty l then ([],[],[]) {- Prekondisi -}
   else
       let (l1,l2,l3) = pecahListGanjilGenap (tail l) in
            {- Memuat l1 dengan bilangan negatif dan 0 -}
           if (head l) <= 0 then (konso (head l) l1, l2, l3)
            {- Memuat l2 dengan bilangan positif dan ganjil -}
           else if (mod (head l) 2) == 1 then (l1, konso (head l) l2, l3)
            {- Memuat l3 dengan bilangan positif dan genap -}
           else (l1, l2, konso (head l) l3)


-- APLIKASI
-- pecahListGanjilGenap [1,3,0,4,-1,4,-9]
```

Soal 7 (Alternate Sort)

Soal ini soal bonus. Kerjakan hanya bila soal-soal sebelumnya sudah selesai dikerjakan.

Nama File: AlternateSort.hs

Header: module AlternateSort where

Diberikan sebuah list, Pak Engi memiliki sebuah algoritma prosedural sebagai berikut.

1. Urutkan list tersebut
2. Bagi list menjadi 2 sama besar, misal l1 dan l2. Jika panjang list ganjil, maka l1 akan memiliki 1 elemen lebih banyak dibanding l2
3. Ambil elemen terkecil dari l1, masukkan ke akhir l3.
4. Ambil elemen terbesar dari l2, masukkan ke akhir l3.
5. Ulangi langkah 3 dan 4 sampai kedua list kosong.

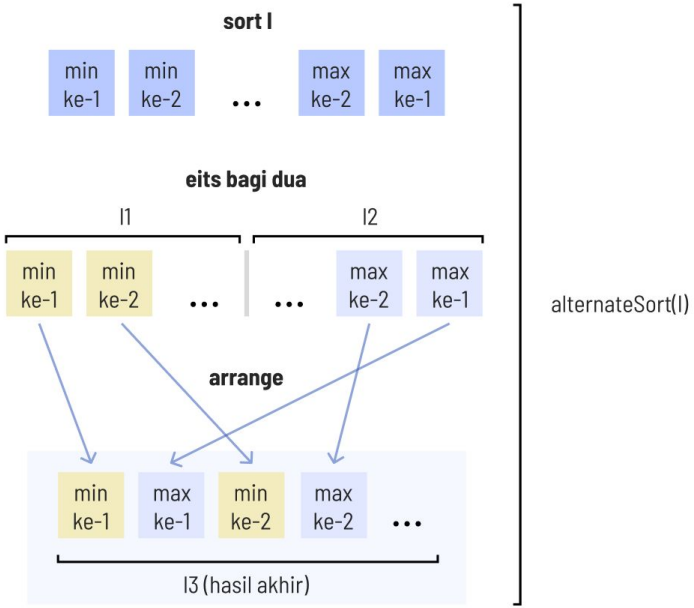
Contohnya, jika list awal adalah [9,10,11,12], maka l3 akan menjadi [9,12,10,11]

Pak Engi telah selesai membuat algoritma prosedural tersebut. Anda, sebagai pemrogram handal, ingin membuat versi fungsional dari kode tersebut. Namun, anda menyadari bahwa langkah prosedural tersebut terlalu kompleks untuk diimplementasikan dalam waktu 2 jam, sehingga anda ingin mencari cara lain untuk mengimplementasikan algoritma tersebut. Buatlah program yang dapat melakukan algoritma tersebut!

Contoh aplikasi fungsi dan hasilnya:

```
> alternateSort[9,10,11,12]
[9,12,10,11]
> alternateSort[5,2,5,2,1]
[1,5,2,5,2]
```

Apa sih yang sebenarnya dilakukan?



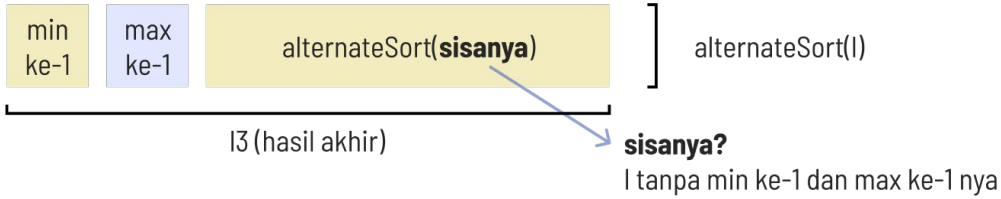
alternateSort(l)

inget hasil akhir, **jangan terpaku sama prosedur!**

Perhatikan bahwa hasil akhir yang diinginkan adalah list yang selang-seling antara nilai maksimum dan minimumnya.

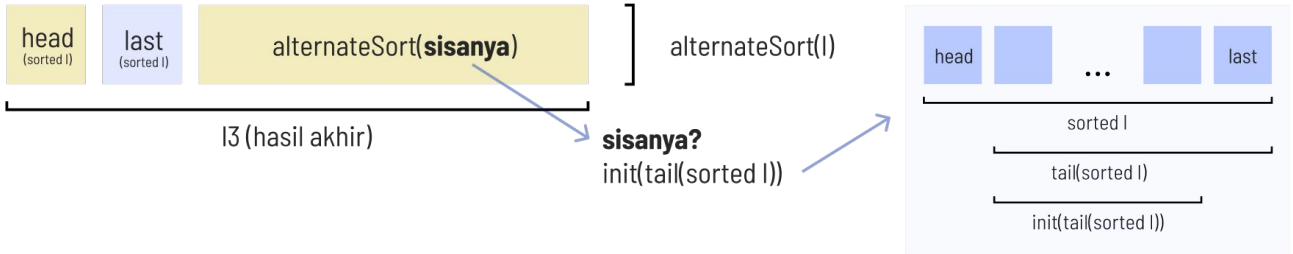
Bagaimana ***cara meng-achieve hasil yang sama*** dengan pendekatan fungsional?

Pendekatan Fungsional (Rekursif)



Gimana mengimplementasikan ini di list rekursif yang cuman ada primitif: head(l), tail(l), init(l), last(l), dll?

Jika kita dapat men-sort l secara ascending, kita bisa dapat:



Kode

```
module AlternateSort where
-- definisikan isEmpty, isOneElmt, dan konso

-- DEFINISI DAN SPESIFIKASI FUNGSI ANTARA
minEl :: [Int] -> Int
{- minEl l menghasilkan nilai terkecil dari l
-}
-- REALISASI
minEl l
  | isOneElmt l = head l
  | head l < minEl (tail l) = head l
  | otherwise = minEl (tail l)

del :: Int -> [Int] -> [Int]
{- del x l menghasilkan elemen l yang bernilai
x yang muncul pertama kali -}
-- REALISASI
del x l
  | isEmpty l = l
  | head l == x = tail l
  | otherwise = konso (head l) (del x (tail l))
```

```
sort :: [Int] -> [Int]
{- sort l menghasilkan list l yang terurut membesar -}
-- REALISASI
sort l
  | isEmpty l = l
  | otherwise = konso (minEl l) (sort (del (minEl l) l))

-- DEFINISI DAN SPESIFIKASI FUNGSI UTAMA
alternateSort :: [Int] -> [Int]
{- alternateSort l menghasilkan list yang sama dengan list
l yang sudah diurutkan, lalu dibagi dua menjadi l1 dan l2,
lalu diambil elemen terkecil dari l1 lalu elemen terbesar
dari l2 hingga kedua list kosong -}
-- REALISASI
alternateSort l
  | isEmpty l || isOneElmt l = l
  | otherwise = konso (head (sort l)) (konso (last (sort
l)) (alternateSort (init(tail(sort l)))))

-- APLIKASI
-- alternateSort [9,10,11,12]
```

Mudah kan? :D
Ada pertanyaan?