0.1 - Tipe-tipe Recursion

0.1.1 - Standard Recursion

Recursion merupakan sebuah konsep dimana sebuah fungsi memanggil dirinya sendiri, recursion banyak dipakai pada bahasa fungsional bersih, namun dapat dipakai juga pada bahasa lain yang bukan bahasa fungsional bersih.

contoh recursion dalam bahasa Haskell, yang akan menghitung factorial :

```
factorial :: Int -> Int
factorial 0 = 1
factorial n = n * factorial (n - 1)
```

dari fungsi diatas kita dapat melihan bila kondisi atau variable n adalah 0 maka akan mengembalikan anggak 1 dan akan keluar dari recursion, dan jika nilai n bukan lah 0 maka fungsi ini akan memanggil dirinya sendiri dengan perubahan pada parameter n, seperti inilah bahasa yang tidak memiliki fungsi loop seperti while, for loop dapat melakukan looping, namun cara ini merupakan cara yang sangat buruk untuk melakukan loop, cara recursion seperti ini merupakan cara yang sangat tidak efisien karna setiap kali fungsi memanggil diri sendiri maka ada kemungkinan memori stack kita penuh, dan dapat terjadi Error stack overflow.

seperti inilah ilustrasinya, jika kita memasukan angka 5 pada input fungsi factorial :

```
factorial 5 = 5 * factorial ( 5 - 1 )
factorial 4 = 4 * factorial ( 4 - 1 )
factorial 3 = 3 * factorial ( 3 - 1 )
factorial 2 = 2 * factorial ( 2 - 1 )
factorial 1 = 1 * factorial ( 1 - 1 )
factorial 0 = 1
-- then
factorial 5 = 5 * 4 * 3 * 2 * 1
```

seperti visualisasi diatas fungsi factorial akan menunggu dan memmanggil dirinya sendiri sampai memenuhi nilai 0, setelah itu akan kembali lagi ke awal, bayangkan saja bila kita memasukan nilai 100 untuk inputnya, maka fungsi factorial akan memanggil 100x dirinya sendiri, dan ada kemungkinan akan terjadi stack ovewflow karna memori stack penuh akibat fungsi factorial

0.1.2 - Tail Recursionn

Tail recursion juga merupakan sebuah tehnik recursion tetapi cara tail recursion melakukan recursion sangat lah berbeda dari recursion biasa, dan juga lebih efisien.

seperti inilah kita melakukan sebuah tail recursion :

```
factorial :: (Eq t, Num t) => t -> t
factorial n = go n 1

go :: (Eq t, Num t) => t -> t -> t
go 1 a = a
go n a = go ( n - 1 ) ( a * n )
```

bisa dilihat kode diatas bahwa kita memerlukan 2 buah fungsi, fungsi untuk memanggil dan fungsi yang akan melakukan perhitungan, jika saya memberikan angka 5 ke dalam fungsi factorial maka seperti inilah kalkulasinya :

```
factorial 5 = go 5 1
            = go 4 5 -- go ( 5 - 1 ) ( 5 * 1 )
            = go 3 20 -- go ( 4 - 1 ) ( 4 * 5 )
            = go 2 60 -- go ( 3 - 1 ) ( 3 * 20 )
            = go 1 120 -- go ( 2 - 1 ) ( 60 * 2 )
```

dari visualisasi diatas kita dapat melihat bahwa fungsi factorial hanya memanggil fungsi go sekali sampai parameter n yang ada di go menjadi 1 dan menggembalikan variable a yaitu 120, maka dalam fungsi go parameter n merupakan parameter tujuan dan parameter a merupakan tempat kita melakukan perhitungan, dengan melakukan recursion seperti ini kita bisa menghindari stack overflow karna kita hanya memanggil 2 fungsi saja.