Responsi

Praktikum 1 - S1

made with love by labpro

Common Mistakes

Tidak di compile di lokal

Kebanyakan praktikan langsung melakukan submit ke olympia tanpa mengetahui programnya dapat dijalankan secara benar atau tidak

Kesalahan sintaks

Dikarenakan sintaks haskell yang baru, banyak praktikan yang tidak mengetahui sintaks yang digunakan oleh haskell

Tidak menuliskan definisi

Walaupun hanya merupakan best practice, mendefinisikan fungsi yang dibuat dapat mempermudah penggambaran soal

Tidak sempat di grade

Praktikan melakukan grading soal secara bersamaan, tidak bertahap. Sehingga terdapat waktu menunggu grading

Common Mistakes

Urutan If-else

Kebanyakan praktikan yang tidak mendapatkan nilai penuh disebabkan oleh urutan If-else yang tidak tepat

Perhatikan Soal

Kebanyakan praktikan tidak membaca soal dengan baik, seperti penamaan modul, permintaan soal, rumus yang digunakan dalam soal, dsb.

Tidak teliti

Kebanyakan praktikan tidak mendapatkan nilai penuh karena tidak teliti dalam penggunaan variabelnya. Ada yang membuat *infinite recursion* dan variabel tertukar dalam kodenya.

Kelengkapan If-else

Beberapa praktikan tidak memiliki case kondisional yang lengkap, sehingga tidak mencakup semua syarat yang ada pada soal

Soal 1 (Luas Trapesium)

Nama File: LuasTrapesium.hs

Header: module LuasTrapesium where

Buatlah definisi, spesifikasi, dan realisasi fungsi **luasTrapesium** yang menerima masukan 3 buah bilangan real (float) t, s1, s2 dengan t = tinggi trapesium, s1 = panjang sisi sejajar 1, dan s2 = panjang sisi sejajar 2 (asumsikan: t > 0, s1 > 0, s2 > 0, dan s1 tidak sama dengan s2) dan menghasilkan luas trapesium berdasarkan rumus: luas = ½ * t * (s1 + s2)

Contoh aplikasi fungsi dan hasilnya:

```
> luasTrapesium 2 4 3
7.0
```

Soal 1 (Luas Trapesium)

```
module LuasTrapesium where

-- DEFINISI DAN SPESIFIKASI
luasTrapesium :: Float -> Float -> Float -> Float
{- luasTrapesium(t,s1,s2) adalah fungsi yang memberikan luas trapesium
dari nilai panjang sisi sejajar 1 dan 2 serta tinggi trapesium -}
-- t, s1, s2 merupakan float (bilangan real)

-- REALISASI
luasTrapesium t s1 s2 = 0.5 * t * (s1 + s2)

-- APLIKASI
  -- luasTrapesium 2 4 3
```

Soal 2 (Konversi Suhu)

Nama File: KonversiSuhu.hs

Header: module KonversiSuhu where

Buatlah sebuah fungsi **konversiSuhu** (definisi, spesifikasi, dan realisasi) yang digunakan untuk mengkonversi suhu dari satu satuan Celcius ke satuan suhu yang lain, yaitu Fahrenheit, Reamur, atau Kelvin. Berikut adalah rumus untuk melakukan konversi jika suhu dalam derajat Celcius adalah C:

Suhu Tujuan	Rumus Konversi
Reamur	4/5 * C
Fahrenheit	(9/5 * C) + 32
Kelvin	C + 273.15

Fungsi ini menerima masukan:

- 1 buah nilai bertipe real (float), misalnya t, yang merupakan besaran suhu dalam derajat Celcius.
- 1 buah kode satuan suhu konversi, bertipe karakter, misalnya k, yang diasumsikan bernilai 'R' (Reamur), 'F' (Fahrenheit), atau 'K' (Kelvin).

Fungsi menghasilkan suhu dalam satuan k yang merupakan konversi suhu t derajat Celcius.

Contoh:

konversiSuhu (25, 'R') artinya adalah konversi suhu 25 derajat Celcius ke suhu dalam derajat Reamur.

Soal 2 (Konversi Suhu)

```
module KonversiSuhu where

-- DEFINISI DAN SPESIFIKASI
konversiSuhu:: Float -> Char -> Float
{- konversiSuhu(s, c) adalah fungsi yang memberikan hasil konversi
suhu dari nilai s dan karakter c -}
-- s merupakan float, c merupakan karakter

-- REALISASI
konversiSuhu s c
      | c == 'R' = s * 4 / 5
      | c == 'F' = s * 9 / 5 + 32
      | c == 'K' = s + 273.15

-- APLIKASI
  -- konversiSuhu 25 'R'
```

Soal 3 (Jam Bangun)

Nama File : JamBangun.hs

Nama Modul : JamBangun

Jane adalah seorang yang sangat teratur, ia selalu tidur **tepat** selama 8 Jam, pada hari ini ia tidur pada pukul **23.45.00**.

Namun, teman sekamar Jane tiba tiba melihat bahwa ada pesan masuk pada ponsel Jane dari dosen pembimbingnya. Jane diminta untuk menelepon dosen tersebut besok pada jam tertentu.
Ada kemungkinan bahwa Jane terlambat bangun dan melewati panggilan yang sangat penting ini sehingga teman Jane harus membangunkan Jane jika Jane akan terlambat bangun.

Bantulah teman Jane untuk membuat program yang dapat menerima input 3 Integer berupa jam(0..23) menit(0..59) detik(0.59) yang merupakan jam yang ditentukan oleh dosen pembimbing Jane
Lalu program dapat mengeluarkan output berupa **tuple** yang berisi:
- Apakah Jane akan bangun melewati jam yang telah ditentukan (True: ya, False: Tidak)
- Waktu selisih antara waktu Jane bangun dan waktu yang ditentukan pak Dosen.

Asumsi: input selalu valid.

Contoh:
```
> jamBangun 07 15 00
(True,0,30,0)
```

Soal 3 (Jam Bangun)

```
module JamBangun where

-- DEFINISI DAN SPESIFIKASI
jamBangun :: Int -> Int -> Int -> (Bool, Int, Int, Int)
{-
      jamBangun(j, m, d) adalah fungsi yang memberikan jawaban terdapat problem statement yang
diberikan.
      Berikut adalah permasalahan yang harus diselesaikan:
      a. Fungsi harus mencari tahu apakah waktu bangun melebihi waktu pada input
      b. Fungsi harus mencari selisih waktu bangun dan input

      !Kunci untuk menjawab jawaban ini adalah bukan membandingkan waktu tidur dengan input tetapi
menambahkan
      waktu tidur dengan 8 jam dan membandingkan waktu tersebut (waktu bangun) dengan input!
-}
-- j, m, dan d adalah Integer
```

Soal 3 (Jam Bangun)

```haskell
-- REALISASI
jamBangun j m d =
      let bangun_seconds = 7*3600 + 45*60
          dosen_seconds = j * 3600 + m*60 + d
          selisih_seconds
            | bangun_seconds > dosen_seconds = bangun_seconds - dosen_seconds
            | otherwise = dosen_seconds - bangun_seconds

          result_jam = div selisih_seconds 3600             -- jam dalam detik
          result_menit = div (mod selisih_seconds 3600) 60  -- menit dalam detik
          result_detik = mod selisih_seconds 60             -- ya detik
      in
      (
            bangun_seconds > dosen_seconds, -- True / False berdasarkan kondisi
            result_jam, result_menit, result_detik
      )
```

Soal 4 (Klasifikasi Komputer)

Nama File : KlasifikasiKomputer.hs

Nama Modul : KlasifikasiKomputer

Sebuah perusahaan pembuat video game ingin melakukan optimisasi pada komputer pengguna yang memiliki game mereka. Optimisasi dilakukan dengan cara mengklasifikasikan komputer pengguna dan mengubah pengaturan game sesuai dengan kelompok kemampuannya. Setiap komputer dapat masuk ke salah satu dari 5 kelompok, yaitu 1, 2, 3, 4, dan 5.
Klasifikasi komputer akan dilakukan berdasarkan kemampuan CPU, kemampuan GPU, dan kemampuan harddisk.
Ketentuan klasifikasinya adalah sebagai berikut:
1. Jika kemampuan CPU lebih dari 7, kemampuan GPU lebih dari 7, dan kemampuan harddisk lebih dari 7, maka termasuk kelompok 5
2. Jika setidaknya satu kemampuan bernilai **kurang dari sama dengan** dari 7, maka termasuk kelompok 4
3. Jika nilai kemampuan CPU, GPU, **dan** harddisk kurang dari sama dengan 7, maka termasuk kelompok 3
4. Jika nilai kemampuan CPU **atau** GPU kurang dari 5, maka termasuk kelompok 2
5. Jika ada **salah satu nilai kemampuan yang bernilai kurang dari 2**, maka termasuk kelompok 1 (tidak peduli dengan nilai kemampuan lainnya)

Prioritas kelompok terurut dari nomor kelompok **terkecil** hingga terbesar.
Misalnya, apabila masukan cocok dengan kelompok 2 dan 4, maka yang dianggap benar adalah kelompok 2.

Contoh:
> klasifikasi 8 9 4
4
> klasifikasi 4 10 2
2
> klasifikasi 6 10 1
1

Klasifikasi sesuai Nomor

1. CPU < 2 atau GPU < 2 atau HDD < 2
2. CPU < 5 atau GPU < 5
3. CPU ≤ 7 dan GPU ≤ 7 dan HDD ≤ 7
4. CPU ≤ 7 atau GPU ≤ 7 atau HDD ≤ 7
5. CPU > 7 dan GPU > 7 dan GPU > 7

Urutan *conditional* berpengaruh, pengecekan untuk kondisi **yang beririsan** diprioritaskan dari kondisi kelompok terkecil.

Aturan untuk nomor 5 bisa **diubah menjadi "else" saja** karena
not(nomor 4)
= not(CPU ≤ 7 atau GPU ≤ 7 atau HDD ≤ 7)
= CPU > 7 dan GPU > 7 dan GPU > 7
= nomor 5

Soal 4 (Klasifikasi Komputer)

```
module KlasifikasiKomputer where

-- DEFINISI DAN SPESIFIKASI
klasifikasi :: Int -> Int -> Int -> Int
{- klasifikasi(cpu, gpu, hdd) mengklasifikasikan komputer berdasarkan kemampuan CPU (cpu), GPU
(gpu), dan hard disk (hdd) -}

-- REALISASI
klasifikasi cpu gpu hdd
  | cpu < 2 || gpu < 2 || hdd < 2 = 1
  | cpu < 5 || gpu < 5 = 2
  | cpu <= 7 && gpu <= 7 && hdd <= 7 = 3
  | cpu <= 7 || gpu <= 7 || hdd <= 7 = 4
  | otherwise = 5

-- APLIKASI
-- klasifikasi 8 9 4
```

Soal 5 (Deret Segitiga)

Time limit	1 s
Memory limit	64 MB

Nama File: DeretSegitiga.hs

Header: module DeretSegitiga where

Buatlah definisi, spesifikasi, dan realisasi dari fungsi **deretSegitiga** yang merupakan fungsi untuk mencari nilai bilangan ke-n pada deret segitiga. Deret segitiga adalah:1, 3, 6, 10, 15, ...

Fungsi ini **harus** diselesaikan menggunakan pendekatan rekursif.

```
deretSegitiga :: Int -> Int
-- deretSegitiga(n) menghasilkan nilai bilangan ke-n pada deret segitiga
-- prekondisi: n > 0
```

Contoh aplikasi fungsi dan hasilnya:

No	Aplikasi	Hasil
1.	deretSegitiga 1	1
2.	deretSegitiga 5	15
3.	deretSegitiga 100	5050

Soal 5 (Deret Segitiga)

```haskell
module DeretSegitiga where

-- DEFINISI DAN SPESIFIKASI
deretSegitiga:: Int-> Int
{- Menghitung suku ke-n dari deret segitiga menggunakan
metode rekursif -}

-- REALISASI
deretSegitiga n
  | n == 1 = 1
  | otherwise = n + deretSegitiga (n-1)

-- APLIKASI
  -- deretSegitiga 100
```

Penjelasan Algoritma

Perhatikan bahwa suku ke-n dalam deretSegitiga adalah dengan **menjumlahkan semua digit dari 1 sampai n** atau bisa disebut **1...n .**

Maka akan digunakan metode rekursif yang berhenti ketika mencapai **basis yaitu angka 1.**

Metode rekursif dilakukan adalah dengan ***decrement* 1 pada nilai n** dan **ditambahkan dengan nilai n saat ini**.

Soal 6 (NbKelipatanX)

Time limit	1 s
Memory limit	64 MB

Nama File: NbKelipatanX.hs

Header: module NbKelipatanX where

Buatlah definisi, spesifikasi, dan realisasi fungsi **nbKelipatanX** yang menerima masukan dua buah integer positif (integer > 0), misalnya **m** dan **n**, serta sebuah integer positif lain, yaitu **x**, dan menghasilkan banyaknya bilangan kelipatan x di antara m dan n (m dan n termasuk) dengan menggunakan ekspresi rekursif.

Bilangan y disebut kelipatan bilangan x, jika y habis dibagi dengan x.

Prekondisi/syarat/asumsi yang berlaku adalah m <= n dan x <= n.

Contoh aplikasi dan hasilnya:

No	Aplikasi	Hasil	Keterangan
1.	nbKelipatanX 1 1 1	1	Kelipatan 1 di antara [1..1] adalah 1
2.	nbKelipatanX 1 10 2	5	Kelipatan 2 di antara [1..10] adalah 2, 4, 6, 8, 10
3.	nbKelipatanX 5 14 3	3	Kelipatan 3 di antara [5..14] adalah 6, 9, 12

Soal 6 (NbKelipatanX)

```
module NbKelipatanX where

-- DEFINISI DAN SPESIFIKASI
nbKelipatanX :: Int-> Int -> Int -> Int
{- menghasilkan banyaknya bilangan kelipatan x di antara m dan n dengan menggunakan ekspresi rekursif -}

-- REALISASI
nbKelipatanX m n x
  | m == n =
      if mod n x == 0 then 1
      else 0
  | mod n x == 0 = 1 + nbKelipatanX m (n - 1) x
  | otherwise = 0 + nbKelipatanX m (n - 1) x

-- APLIKASI
  -- nbKelipatanX 1 1 1 -> 1
  -- nbKelipatanX 1 10 2 -> 5
  -- nbKelipatanX 5 14 3 -> 3
```

Banyak kesalahan:

```
| m == n && n == x = 1
| m == n && n /= x = 0
| mod m x == 0 = 1 + (nbKelipatanX (m + 1) n x)
| otherwise = nbKelipatanX (m + 1) n x
```

Tidak mempertimbangkan jika n atau m merupakan kelipatan x, sehingga kehilangan 1 buah kelipatan, yaitu m atau n sebagai kelipatan x

Soal 7 (Hitung Bensin)

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

Nama File : HitungBensin.hs

Nama Modul : HitungBensin

Setelah setahun tidak pulang kampung, akhirnya Tuan Vin pun memberanikan diri untuk meminta cuti kepada bosnya. Bosnya sebenarnya ingin langsung menyetujui cuti Tuan Vin. Akan tetapi, dia ingin Tuan Vin membereskan pekerjaannya terlebih dahulu. Pekerjaan Tuan Vin sebenarnya cukup mudah. Dia hanya perlu menyiapkan bensin untuk seluruh kendaraan perusahaannya.

Kendaraan perusahaan Tuan Vin memiliki rute yang sangat unik. Awalnya, kendaraan tersebut akan terletak pada posisi X. Kemudian jika X adalah bilangan genap, kendaraan tersebut akan bergerak ke titik X/2. Jika X adalah bilangan ganjil, kendaraan tersebut akan bergerak ke posisi (3X + 1). Hal ini terus dilakukan sampai kendaraan tersebut sampai ke kantor pusat yang terletak pada posisi 1. Untuk setiap perpindahan posisi, kendaraan tersebut akan menghabiskan bensin sebanyak 1 unit. Jika pada awalnya suatu kendaraan terletak pada posisi 11, kendaraan tersebut akan berpindah ke (11*3 + 1) = 34. Kemudian, kendaraan tersebut kemudian berpindah ke posisi 17,52,26,13,40,20,10,5,16,8,4,2 dan berakhir pada posisi 1 sehingga kendaraan tersebut menghabiskan bensin sebanyak 14 unit.

Tiap harinya, akan ada kendaraan yang berangkat dari posisi A sampai dengan posisi B. Tuan Vin kemudian menjumlahkan banyaknya bensin yang dibutuhkan untuk tiap-tiap kendaraan dari posisi A sampai dengan posisi B. Tuan Vin takut dia tidak sempat menyelesaikan kalkulasinya sebelum hari cutinya tiba. Sebagai teman baik Tuan Vin, Anda pun ingin membantu Tuan Vin dengan membuat sebuah fungsi untuk menghitung bensin yang perlu disiapkan. Fungsi **hitungBensin** menerima 2 buah bilangan bulat, A dan B (A <= B). Fungsi ini kemudian mengeluarkan sebuah bilangan bulat yang menunjukkan konsumsi bensin dari tiap-tiap kendaraan dari A sampai B. Tentu saja sebagai programmer yang baik, Anda harus membuat definisi, spesifikasi, dan realisasi fungsi ini.

Contoh :

> hitungBensin 11 11

14

> hitungBensin 1 10

67

Keterangan:

Pada contoh kedua, 67 didapatkan dengan menjumlahkan bensin yang diperlukan untuk mobil yang mulai pada posisi ke-1, posisi ke-2, posisi ke-3, hingga posisi ke-10

Soal 7 (Hitung Bensin)

```haskell
module HitungBensin where

-- DEFINISI DAN SPESIFIKASI
hitungBensin :: Int -> Int -> Int
{-
    Menghitung hasil jumlah hitungBensin1Rute A, hitungBensin1Rute A + 1, ..., hitungBensin1Rute B
-}

hitungBensin1Rute :: Int -> Int
{-
    Menghitung total bensin yang dibutuhkan untuk bergerak dari posisi X ke posisi 1, dengan
    aturan sebagai berikut:
    Keterangan: N adalah posisi saat ini
    1. Apabila N tidak habis dibagi 2 maka berpindah posisi ke (3 * N) + 1
    2. Apabila N habis dibagi 2 maka berpindah posisi ke N / 2
-}
```

Soal 7 (Hitung Bensin)

```
-- REALISASI
hitungBensin1Rute n
    | n == 1 = 0
    | (mod n 2) == 0 = 1 + (hitungBensin1Rute (div n 2))
    | otherwise = 1 + (hitungBensin1Rute (3 * n + 1))

hitungBensin a b
    | a == b = hitungBensin1Rute a
    | otherwise = hitungBensin1Rute a + (hitungBensin (a + 1) b)


-- APLIKASI
-- hitungBensin 1 10
```

Mudah kan? :D
Ada pertanyaan?