Започнат на	Sunday, 16 January 2022, 11:30
Състояние	Завършен
Приключен на	Sunday, 16 January 2022, 12:22
Изминало време	52 мин. 34 сек.
Оценка	Още не е оценен

Въпрос **1**

Отговорен

От максимално 1,00

Обяснете понятието „дефиниране на функция на функционално ниво“ в езика Haskell. Дайте поне два пример за дефиниция на функция на функционално ниво.

Дефинирането на функция на функционално ниво предполага нейното действие да се опише чрез връзката ѝ с други функции, а не като резултат от прилагането ѝ върху коректно избрано множество от аргументи.

1. Ако искаме да направим композиция от фунцкии - f(g(s(x))), то тогава можем да напишем следната функция:
   functionComposition :: (a -> a) -> (a -> a) -> (a -> a) -> (a -> a)
   functionComposition f g s = f.g.s
2. Ако искаме първо да намерим абсолютните стойности на числа в списък и след това да ги повдигнем на трета степен, функцията би изглеждала по следния начин:
   cubeAfterAbs = (^3).abs
   Тази функция можем да приложим върху списък от числа по следния начин : map cubeAfterAbs xs, където xs е списък от тип [Int]

Въпрос **2**

Отговорен

От максимално 1,00

Кои от следните конструкции са коректно дефинирани (валидни) списъци в Haskell? Обосновете отговорите си. Посочете типовете на валидните списъци.

(а) [(”Hello”,42),(”Hello”,”42”)]

(б) [[”Hello”,”My”],[”Hello”,”My”,”World”]]

(в) [[1,2],[3,4],[5]]

(г) [[[3]],[[2,1]],[]]

а - невалиден, защото типовете на елементите не са еднородни. (”Hello”,42) има тип (String, Int), а (”Hello”,”42”) - (String, String**)**
б - валиден списък, защото всички негови елементи са от тип [String], така че целият списък има тип [[String]]

в - валиден списък, защото всички негови елементи са от тип [Int], така че целият списък има тип [[Int]]

г - ввалиден списък, защото първите му два елемента са от тип [[Int]], а третият - празният списък - принадлежи към всеки списъчен тип. Така типът на целия списък е [[[Int]]]

Въпрос **3**

Отговорен

От максимално 1,00

Как се конструира списък чрез техниката на определяне на неговия обхват (list comprehension) в езика Haskell? Обяснете общия случай и дайте поне два примера, в които се използват различни възможности на тази техника (включително напишете получените резултати).

Общият синтаксис на конструирането на списък, използвайки техниката на определяне на неговия обхват (list comprehension) е следният:
[expression | q1..qn], където expression е израз, а за qi, i в интервала [1...n], имаме следните възможости:

- генератор p <- lExpression, където p е образец, а lExpression е израз от списъчен тип
- филтър (незадължителен компонент) от вида bExpression (булев израз)

В qi могат да бъдат използвани всички q1, ..., q i-1.

Генераторът е списъкът, от който ще извличаме елементите на новия списък, който конструираме, а филтърът е това, което ще ни казва кои точно елементи от генератора да включим в новопостроения списък.

Пример с филтър:
[x | x <- [1..10], mod x 3 == 0] - [3,6,9] - т.е. извличаме тези елементи от списъка [1..10] , които се делят на 3
Пример без филтър, но пък имаме промяна на елемента в новоконструирания списък
[4*x | x <- [1..5]] - [4,8,12,16,20] - т.е. в новия списък включваме елементите от генератора, умножени по 4

Въпрос **4**

Отговорен

От максимално 1,00

Дайте пример за дефиниция на функция на Haskell, в която се използва обща (а не примитивна) рекурсия върху списъци. Обяснете каква задача се решава с помощта на тази функция.

Нека имаме фунцкията pair, която ни връща двойки от елементи от два списъка. Тя се дефинира с обща рекурсия върху списъци по следния начин:
pair :: [Int] -> [Int] -> [(Int, Int)]

pair [] [] = [] - тук и двата списъка са празни, следователно връщаме празен списък

pair [] _ = [] - ако първият списък е празен, то няма как да генерираме повече двойки и отново връщаме празен списък

pair _ [] = [] - аналогично на горното, но за втория списък

pair (x:xs) (y:ys) = (x,y) : pair xs ys
Последният ред е общият случай, в който взимаме първите елементи на двата списъка и правим една двойка от тях. Фунцкията ще прави такива двойки от числа, докато дължините на списъците са равни. Ако списък 1 има дължина 3, а списък 2 има дължина 8, то в резултата ние ще имаме само 3 двойки.

Въпрос **5**

Отговорен

От максимално 1,00

Обяснете понятието „образец“ (pattern) в езика Haskell. Дайте примери за поне три типа образци и обяснете кога те се съпоставят успешно със съответните аргументи.

Образецът в Haskell може да бъде:

- литерал (2,False, 'a') - за да бъде успешно съпоставен с този образец, един аргумент трябва да има стойност, равна на тази на образеца
- име на променлива (x) - аргумент с произволна стойност се съпоставя успешно с този образец
- символ wildcard ("_") - този символ ни позволява да съпоставяме безусловно аргументи с образеца, т.е. успешно се съпоставя с аргументи от произволен тип и произволна стойност
- вектор (p1, ... pn) - съпоставя се успешно с аргумент от тип (v1, ..vn), където всяко pi се съпоставя успешно със съответното си qi

Въпрос **6**

Отговорен

От максимално 1,00

Проследете стъпка по стъпка процеса на оценяване и напишете оценката на всеки от следващите изрази на езика Haskell:

(\x y z -> x y z) (\x y -> x*x+y-1) 2 3

product (map (\f -> f 2) [\x->x, \x->x*x, \x->x*x*x])

(\x y z -> x y z) (\x y -> x*x+y-1) 2 3

1. (\x y -> x*x+y-1) 2 3 - на x се присвоява стойност 2, а на y - 3, резултатът ще е 4 + 3 - 1 = 6
2. (\x y z -> x y z) - тук на y се присвоява стойността 6, получена на горния ред, y е 2, z е 3, следователно целия израз ще върне 6 2 3

product (map (\f -> f 2) [\x->x, \x->x*x, \x->x*x*x])

1. map (\f -> f 2) [\x->x, \x->x*x, \x->x*x*x] - на x се присвоява стойността 2, тоест списъкът, подаден като аргумент на map ще изглежда по следния начин
   [2, 4, 8]
2. след това върху списъка [2, 4, 8] прилагаме фунцкията product, която намира произведнието на всички елементи в даден списък, т.е. крайният резултат ще е 2*4*8 = 64

Въпрос **7**

Отговорен

От максимално 1,00

Обяснете понятието „алгебричен тип“ в езика Haskell. Дайте пример за дефиниция на полиморфен алгебричен тип.

Алгебричният тип в Haskell се дефинира като се започне със запазената в езика дума data, последвана от името на алгебричния тип, изписано с главна буква, знак за равенство и конструктурите на този алгебричен тип. Всеки от конструктурите трябва да започва с главна буква също.
Един от най-елементарните алгебрични типове е този, в който конструкторите му просто изброят елементите на типа.

Пример за такъв е следният алгебричен тип:

data Color = Blue | Red | Purple | Green

Пример за полиморфен алгебричен тип е двоичното дърво, дефинирано по следния начин:
data BTree a = Nil | a (BTree a) (BTree a)

където Nil е конструкторът за празно дърво, а типът a е това, което прави алгебричния тип полиморфен - можем да имаме дърво от тип Int, Char и т.н.

Въпрос **8**

Отговорен

От максимално 1,00

Как се дефинират класове в Haskell? Опишете общия случай и дайте конкретен пример за дефиниция на клас.

Класовете в Haskell представляват колеция от типове, за които имаме додефинирани операции, наречени методи. Всеки тип, принадлежащ на даден клас се нарича негов екземпляр. Всеки екземпляр трябва да изпълнява ограниченията, зададени при дефинирането на класа.
Общ случай:
class <Име> a where
<изброяват се методите на класа>
Клас Eq
class Eq a where

(==) :: a -> a -> Bool

◄ Второ контролно върху задачи - предаване на решения (4та група)

Отиди на ...

Второ контролно по ФП - теоретична част ►