Примерно решение на
"Контролно упражнение № 1 по Функционално програмиране (примерен вариант)"

1. Дайте примери за поне три типа образци (patterns) в езика Haskell. Обяснете кога съответните аргументи са съпоставими с тези образци.
   x - всяка стойност на аргумнта се съпоставя с този образец
   [] - само празен списък е съвместим с този образец.
   [(0, koef):xs] - съпоставя се списък с поне един елемент и главата на списъка е вектор, на който първият елемент е равен на 0, а вторият е с произволна стойност(на koef се съпоставя всяка стойност).
2. Дайте пример за дефиниция на функция, в която се използва обща рекурсия върху списъци:
```
sumList :: [Int] -> Int
sumList [] = 0
sumList (x:xs) = x + sumList xs
```
3. Кои от следните конструкции са коректно дефинирани списъци в Haskell?
   a. ["A",'A'] - **не е коректно дефиниран списък**
   b. 'A' : [] - **коректно дефиниран списък**
   c. [(123,"Hello"),(123,"Hello","World")] - **не е коректно дефиниран списък**
   d. [["123","Hello"],["123","Hello","World"]] - **коректно дефиниран списък**
4. Кои от следните конструкции са коректно дефинирани(валидни) списъци в Haskell? Запишете валидните списъци с помоща на ":" нотацията.
   [1,2,3,[]] - **не е коректна дефиниция на списък - елементите на сисъка не са от един и същ тип. 1,2,3 са цели числа, а последният елемент е списък.**
   [1,[2,3],4] - **не е коректна дефиниция на списък - елементите на сисъка не са от един и същ тип. 1 и 4 са цели числа, а елементът [2,3] е списък.**
   [[1,2,3],[]] - **коректна дефиниция на списък - (1:2:3:[]):([]:[])**
5. Какъв е типа на конструкцията ("Worl",'d',"123",456)?
   ([Char], Char, [Char], Int) - вектор с елементи от тип [Char], Char, [Char], Int
6. Нека е даден списък **lst** от тип [([Char],Int,Float)], съдържащ данни за имената, факултетните номера, и успеха на студентите от една група. Напишете изрази на Haskell, с помоща на които се получават:
   a. списък от имената на студентите от групата: **[name | (name, _, _) <- lst]**
   b. броят на отличните студенти в групата:
   length [mark | (_, _, mark) <- lst, mark == 6.0]
7. Попълнете липсващите изрази в дефиницията **merge** при условие, че тя слива два списъка от цели числа **l1** и **l2**, които са сортирани във възходящ ред(резултатът също е сортиран във възходящ ред и съвпадащите елемени на **l1** и **l2** участват в него в толкова екзумпляра, колкото пъти се срещат сумарно в **l1** и **l2**).
```
merge :: [Int] -> [Int] -> [Int]
merge [] l2 = l2
merge l1 [] = l1
merge (l1:ls1) (l2:ls2)
 | l1 <=l2 = l1 : merge ls1 (l2:ls2)
 | otherwise = l2 : merge ls2 (l1:ls1)
```
8. Дефинирайте функция на Haskell, която премахва всички кратни на 3 числа от даден списък от тип [Int]
```
myFilterFunc :: [Int] -> [Int]
myFilterFunc lst = [x | x <- lst, x `mod` 3 /= 0]
```
9. Дефинирайте функция на Haskell, която проверява дали поредните елементи на списъка от числа **lst** образуват монотонно разстяща редица. Пример за такава редица е **[1,5,6,6,10]**.
```
isMonRast :: [Int] -> Bool
isMonRast [] = True
isMonRast (_:[]) = True
```

```
isMonRast (l1:(l2:ls2))
 |l1 <= l2 = isMonRast (l2:ls2)
 |otherwise = False
```

10. Дефинирайте функция на Haskell, която намира сумата на два полинома, зададени във вида [(<коеф.$_1$>, <степен$_1$>), (<коеф.$_2$>, <степен$_2$>), ... ,(<коеф.$_n$>, <степен$_n$>)], където коефициентите са различни от нула реални числа и степенните показатели са подредени в строго намаляващ ред (резултатът трябва да има същия вид).

```
type Polinom = [(Float, Int)]

polinomSum :: Polinom -> Polinom -> Polinom
polinomSum [] [] = []
polinomSum l1 []  = l1
polinomSum [] l2  = l2
polinomSum ((koef1, stepen1):ls1) ((koef2, stepen2):ls2)
 | stepen1 == stepen2 = if koef1 /= -koef2 then (koef1 + koef2, stepen1) : polinomSum ls1 ls2 else
polinomSum ls1 ls2
 | stepen1 > stepen2  = (koef1, stepen1) : polinomSum ls1 ((koef2, stepen2):ls2)
 | otherwise          = (koef2, stepen2) : polinomSum ((koef1, stepen1):ls1) ls2
```