Контролна работа № 2 по Функционално програмиране
Специалност „Информационни системи“, I курс, 09.06.2022 г.

Задача 1

Да се дефинира **на функционално ниво** функция **`filterTypical :: [String] -> [String]`**, която приема списък от низове като аргумент и връща филтриран списък, който не съдържа „типични думи“. Типична дума е всеки символен низ в предварително попълнен списък `typical`. Елементите на върнатия списък трябва да са в същия ред като в дадения списък.

Примери:

```
typical = ["African", "Roman Tufted", "Toulouse", "Pilgrim",
"Steinbacher"]

filterTypical ["Mallard", "Hook Bill", "African", "Crested",
"Pilgrim", "Toulouse", "Blue Swedish"] → ["Mallard", "Hook Bill",
"Crested", "Blue Swedish"]
filterTypical ["Mallard", "Barbary", "Hook Bill", "Blue Swedish",
"Crested"] → ["Mallard", "Barbary", "Hook Bill", "Blue Swedish",
"Crested"]
filterTypical ["African", "Roman Tufted", "Toulouse", "Pilgrim",
"Steinbacher"] → []
```

Задача 2

Посещаемостта на ученик по предмет може да се представи чрез списък с елементи от алгебричен тип, който определя дали студент е пропуснал часа (`Absent`), закъснял е за часа (`Late`) или е присъствал в часа (`Present`). За да може да получи срочна оценка, ученикът не може да пропусне повече от `n` часа и не може да закъснее повече от `k` пъти подред. Да се дефинира функция от по-висок ред **`canPass :: Criterion -> (StudentRecord -> Bool)`**, която приема двуелементен вектор, представящ максималния разрешен брой пропуснати часове и поредни закъснения, които може да има даден ученик, и връща функция, която за дадена посещаемост на ученик връща булева стойност, определяща дали този ученик може да получи срочна оценка.

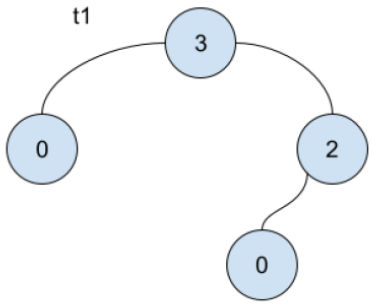
Примери:

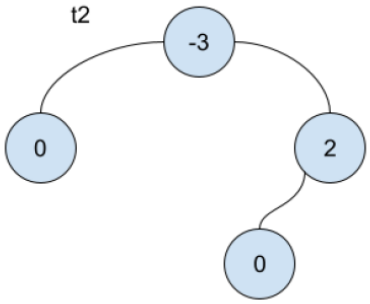
```
cP = canPass (1,2)

type Misses = Int
type Lates = Int
type Criterion = (Misses, Lates)

data Attendance = Absent | Late | Present
type StudentRecord = [Attendance]

cP [Present, Late, Present, Absent, Present, Present, Present,
Absent] → False
cP [Present, Late, Present, Late, Present, Late, Present, Absent,
Late, Present] → True
cP [Present, Late, Present, Late, Late, Late, Present, Present,
Absent, Present] → False
```

Задача 3

Дефиниран е полиморфен алгебричен тип **`BTree a`**, описващ двоично дърво:
`data BTree a = NullT | Node a (BTree a) (BTree a)`.
Да се дефинира функция **`maxSumSubT :: (Ord a, Num a) => BTree a -> a`**, която по подадено двоично дърво намира максималната сума на поддърво на това дърво. Сума на дърво е сумата от всички стойности на възли в дървото. Сумата на празното дърво е `0`. Счита се, че всяко дърво е свое поддърво.

Примери:



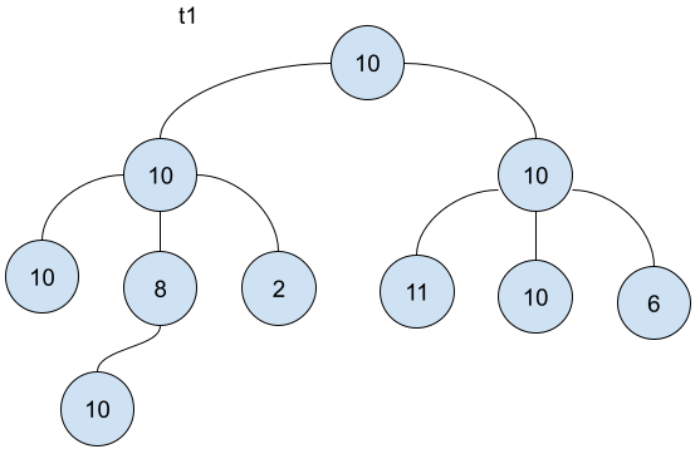


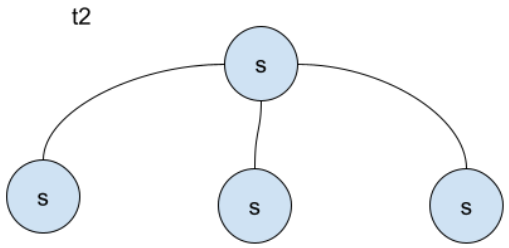

```
maxSumSubT t1 → 5
maxSumSubT t2 → 2
```

Задача 4

Дефиниран е полиморфен алгебричен тип **`NTree a`**, описващ дърво с произволен брой наследници: **`data NTree a = NullT | Node a [(NTree a)]`**.
Скучно дърво е дърво, чиито възли съдържат само една стойност. Да се дефинира функция **`isBoring :: (Eq a) => NTree a -> Bool`**, която за дадено дърво с произволен брой наследници определя дали то е скучно дърво. Празното дърво се приема за скучно дърво.

Примери:






```
isBoring t1 → False
isBoring t2 → True
```