Klausurvorbereitung

Aufgabe 1:
Gegeben sei die folgende grundlegende Definition einer Liste von Integern (IntList). In der Datenstruktur werden Zahlen (Integer) als verkettete Liste gespeichert. Sie enthält die Operationen isEmpty, head, tail, prefix, map und flatMap mit den folgenden Implementierungen:

```
abstract class IntList{

  def isEmpty:Boolean
  def head:Int
  def tail:IntList
  def prefix(elem:IntList):IntList
  def flip:IntList
  def changeNumber(pred:Int=>Boolean, change: Int=>IntList):IntList
}

case object Empty extends IntList{

  def isEmpty = true
  def head= throw new Error ("List is Empty")
  def tail= throw new Error ("List is Empty")
  def prefix(elem:IntList):IntList= Empty
  def flip:IntList= ???
  def changeNumber(pred:Int=>Boolean, change: Int=>IntList):IntList= ???

}


case class Cons(head:Int, tail:IntList) extends IntList{

  def isEmpty= false
  def prefix(elem:IntList):IntList= elem match {

    case Empty => this
    case Cons(h,t) => Cons(h, prefix(t))
   }
  def flip:IntList=???

  def changeNumber(pred:Int=>Boolean, change: Int=>IntList):IntList= ???
}
```

a) Implementieren Sie die Funktion flip, die die Liste von Zahlen umdreht.
b) Entwickeln Sie die Funktion changeNumber mit der folgenden Signatur:
def changeNumber(pred:Int=>Boolean, change: Int=>IntList):IntList. Sie soll die Liste von vorne nach hinten durchlaufen und bei jedem Element überprüfen, ob das übergebene Prädikat pred erfüllt ist. Ist das Prädikat nicht erfüllt, soll das aktuelle Element in die Ergebnisliste übernommen werden, ist es erfüllt, soll nicht das Element übernommen werden sondern das Ergebnis der Anwendung der übergebenen Funktion change auf das Element. Ergebnis ist also ebenfalls eine IntList: z.B.
Cons(1,Cons(2,Cons(3,Empty))).changeNumber(_%2==0, x=>Cons(x,Cons(x,Empty)))
wird
zu Cons(1,Cons(2,Cons(2, Cons(3,Empty)))).

Implementieren Sie die Funktion so, dass die Liste nicht umgedreht wird muss! Wird eine leere Liste übergeben, so soll eine leere Liste herauskommen. Fomulieren Sie die Funktion ausschließlich als Rekursion und wenden Sie keine Higher Order Functions wie filter, map oder flatMap an.

Aufgabe 2:
In Entenhausen wollen nach der üppigen Weihnachtszeit, einige Bewohner ihr Gewicht reduzieren, in dem Sie dIhre Kalorienaufnahme pro Tag verringern. Dazu führen sie ein Tagebuch, bei dem sie aufschreiben, zu welcher Mahlzeit sie wie viele Kalorien zu sich genommen haben. Der entstandene Datensatz hat die folgende Struktur:

```
val calories: List[(String, String, List[(String, Int)])] =
List(("Donald Duck", "2022-01-01",List(("Frühstück",800), ("Mittag",
700), ("Snack",200), ("Abendbrot", 500))), ("Donald Duck", "2022-01-
02",List(("Frühstück",700), ("Mittag", 650), ("Abendbrot", 520))),
("Donald Duck", "2022-01-03",List(("Frühstück",800), ("Mittag", 700),
("Snack",200), ("Abendbrot", 500), ("Snack",150))),("Donald Duck",
"2022-01-04",List(("Frühstück",850), ("Mittag", 900), ("Snack",500),
("Snack", 400))),("Donald Duck", "2022-01-05",List(("Frühstück",600),
("Mittag", 700), ("Snack",200), ("Abendbrot", 100))), ("Dagobert Duck",
"2022-01-01",List(("Frühstück",300), ("Mittag", 500), ("Snack",100),
("Abendbrot", 200))), ("Dagobert Duck", "2022-01-02",
List( ("Frühstück",200), ("Mittag", 300), ("Snack",400), ("Abendbrot",
200))), ("Dagobert Duck", "2022-01-03",List(("Frühstück",800),
("Mittag", 700), ("Snack",200), ("Snack", 200))), ("Dagobert Duck",
"2022-01-04",List(("Frühstück",200), ("Mittag", 300), ("Snack",200),
("Snack", 500))), ("Dagobert Duck", "2022-01-05",List(("Frühstück",200),
("Mittag", 700), ("Abendbrot", 500))))
```

Für jeden Tag gibt es einen Eintrag der folgenden Struktur: (Person, Tag, Liste der Mahlzeiten). Die Liste der Mahlzeiten besteht aus Tupeln mit den Namen der Mahlzeit (Frühstück, Mittag, Abendbrot, Snack) sowie der aufgenommenen Kalorien.

Schreiben Sie folgende Funktionen zur Extraktion von Informationen. Für die Implementierung können sie die folgenden Scala List/Map-Funktionen nutzen:

def map[B](f: (A) ⇒ B): List[B]
def filter(p: (A) ⇒ Boolean): List[A]
def flatMap[B](f: (A) ⇒ GenTraversableOnce[B]): List[B]
def foldLeft[B](z: B)(op: (B, A) ⇒ B): B
def fold[A1 >: A](z: A1)(op: (A1, A1) ⇒ A1): A1
def reduce[A1 >: A](op: (A1, A1) ⇒ A1): A1

Benutzen Sie keine Elemente der imperativen Programmierung wie veränderliche Variablen!

a) Schreiben Sie eine Funktion `dayWithMaxCalories(l:List[(String, String, List[(String, Int)])]):(String,String,Int)`, die ermittelt, von wem an welchem Tag die maximale Tagesmenge an Kalorien aufgenommen wurden. Ergebnis soll ein Tripel sein, dessen erstes Element der Name, das Zweite der Tag und das Dritte die Tageskalorienzahl ist. Gibt es mehrere Einträge mit der gleichen maximalen Tageskalorienzahl, so kann eine beliebige zurück gegeben werden.

b) Schreiben Sie die Funktion `caloriesByMeal(l:List[(String, String, List[(String, Int)])]):Map[String,Int]`. Sie soll berechnen, wie viel Kalorien insgesamt von allen bei den einzelnen angegebenen Mahlzeiten aufgenommen werden. Ergebnis soll eine Map sein, dessen Schlüssel die Mahlzeit und dessen Wert die Kalorien sind. Verwenden Sie nicht den Gruppierungsoperator für die Berechnung sondern aggregieren Sie die Map über fold oder foldLeft.
c) Schreiben Sie die Funktion `caloriesByPerson(l:List[(String, String, List[(String, Int)])]):Map[String,Int]`. Sie soll berechnen, wie viel Kalorien insgesamt von den einzelnen Personen aufgenommen wurden. Ergebnis soll eine Map sein, dessen Schlüssel die Mahlzeit und dessen Wert die Kalorien sind. Verwenden Sie nicht den Gruppierungsoperator für die Berechnung sondern aggregieren Sie die Map über fold oder foldLeft.

Aufgabe 3:
Gegeben sei die folgende Map[String,List[Int]]:
val m= Map(1->List(2,3,4,5), 2->List(1,2,3), 3→List(4,5))
Berechnen Sie für alle map-Einträge die Durchschnittswerte der Listen. Verwenden Sie dafür die map-Funktion.