Συναρτησιακός Προγραμματισμός 2008
Λύσεις στο Πρώτο Φύλλο Ασκήσεων

1. Χρησιμοποιείστε λ-εκφράσεις και τη συνάρτηση `sumf` στις Σημ.1, Ενότ.3.5, για να εκφράσετε το $\sum_{i=1}^{10} \sum_{j=1}^{i} (i+j)^2$, χωρίς να ορίσετε καινούργιες συναρτήσεις όπως οι `outer` και `inner`.

   §

   ```
   sumf (\i->sumf (\j->(i+j)^2) i) 10
   ```

2. Χρησιμοποιώντας την `sumf`, φτιάξτε μία συνάρτηση τύπου `[[Int]]->Int` που να παίρνει μία λίστα με λίστες και να αθροίζει τα μεγέθη τους.

   §

   ```
   sumf (\i->length (l!!(i-1))) (length l)
   ```

3. Φτιάξτε μία συνάρτηση που να παίρνει μία λίστα ακεραίων και να απαλοίφει όλους τους ακέραιους που δεν εμφανίζονται για πρώτη φορά. Για παράδειγμα, η συνάρτησή σας παίρνει τη λίστα `[1,1,0,3,4,3,1]` και επιστρέφει `[1,0,3,4]`.

   § Για κάθε στοιχείο `x` που βρίσκουμε, χρειάζεται να φιλτράρουμε το υπόλοιπο της λίστας με τη συνθήκη `x/=`. Αυτό μας δίνει τον εξής κώδικα:

   ```
   unique :: [Int]->[Int]
   unique [] = []
   unique (x:xs) = x:unique(filter (x/=) xs)
   ```

4. Μία μαθηματική σειρά που προσεγγίζει το συνημίτονο ενός αριθμού x δίνεται από:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$$

(όπου το x εκφράζεται σε ακτίνια). Φτιάξτε μία συνάρτηση

```
cosapprox :: Float->[Float]
```

που να παίρνει έναν πραγματικό αριθμό `x` και να επιστρέφει μία άπειρη λίστα προσεγγίσεων του `cos x` με βάση τον παραπάνω τύπο. Λόγω του σφάλματος που εισάγουν οι υπολογισμοί σε τιμές `Float`, δεν απαιτείται ο υπολογισμός με ακρίβεια μεγαλύτερη από 10^{-7}, ενώ μπορείτε να θεωρήσετε ότι η είσοδος είναι ένας πραγματικός αριθμός μεταξύ `-pi` και `pi`.

§ Θα φτιάξουμε πρώτα την άπειρη σειρά των προσθετέων $(-)x^i/i!$ και θα την πούμε `cosdiffapprox`. Κάθε όρος της σειράς παράγεται εύκολα από τον προηγούμενο ως εξής:

```
-prev*(x^2) / (i*(i+1))
```

όπου `prev` είναι ο προηγούμενος όρος και `i+1` είναι ο αριθμός του οποίου το παραγοντικό μπαίνει στον τρέχοντα παρανομαστή. Επειδή είναι το `x^2` και όχι το `x` που χρειαζόμαστε, θα το υπολογίσουμε μία φορά και θα το ονομάσουμε `xx`. Η παραπάνω έκφραση γίνεται:

```
-prev*xx / (i*(i+1))
```

Προσέχουμε τώρα ότι το `i` είναι ακέραιος. Επίσης, για λόγους καλύτερης ακρίβειας, είναι καλύτερα να κάνουμε πολλαπλασιασμούς και διαιρέσεις εναλλάξ. Η έκφρασή μας γίνεται:

```
-prev / fromIntegral i * xx / fromIntegral (i+1)
```

Για να φτιάξουμε τη σειρά `cosdiffapprox`, χρησιμοποιούμε μία βοηθητική συνάρτηση `diffapproxaux` η οποία παίρνει σαν παραμέτρους αυτά ακριβώς που χρειαζόμαστε: το `xx`, το `prev` και το `i`, και δημιουργεί τη σειρά από το τρέχον σημείο και πέρα:

```
diffapproxaux :: Float->Float->Int->[Float]
diffapproxaux xx prev i =
  current : diffapproxaux xx current (i+2)
  where
  current = -prev / fromIntegral i * xx / fromIntegral (i+1)
```

Προσέξτε πώς η `diffapprox` δεν υπολογίζει μόνο τον τρέχοντα όρο `current`, αλλά καλεί τον εαυτό της για τον επόμενο όρο, περνόντας τις κατάλληλες παραμέτρους: το `xx`, τον νέο όρο `current` και το `i` αυξημένο κατά 2. Ο όρος `current` ονομάζεται μέσα σε `where` και για αναγνωσιμότητα και επειδή χρησιμοποιείται δύο φορές.

Τώρα μπορούμε να ορίσουμε την `cosdiffapprox` ως εξής: παίρνουμε την παράμετρο `x`, την υψώνουμε στο τετράγωνο και, ξεκινώντας από `1.0`, εφαρμόζουμε την `diffapproxaux` με τις κατάλληλες παραμέτρους:

```
cosdiffapprox :: Float->[Float]
cosdiffapprox x = 1.0:diffapproxaux (x^2) 1.0 1
  where
  diffapproxaux :: Float->Float->Int->[Float]
  diffapproxaux xx prev i =
    current : diffapproxaux xx current (i+2)
    where
    current = -prev / fromIntegral i * xx / fromIntegral (i+1)
```

Προσέχουμε τώρα ότι, κάθε στοιχείο `j` της `cosapprox` είναι το άθροισμα όλων των στοιχείων από `0` έως `j` της `cosdiffapprox`. Θα φτιάξουμε μία συνάρτηση `sums` που να παίρνει μία (άπειρη) λίστα `d` και να παράγει την (άπειρη) λίστα των αθροισμάτων των στοιχείων της `d`. Θα χρησιμοποιήσουμε μία βοηθητική συνάρτηση `sumsaux` που συσσωρεύει ένα άθροισμα `s` που θα το περνάει στον εαυτό της για τον υπολογισμό του επόμενου στοιχείου:

```
sumsaux s (x:xs) = newS : sumsaux newS xs  where newS = s+x
```

Η πρώτη κλήση της συνάρτησης αυτής είναι φυσικά με συσσωρευτικό άθροισμα `0`:

```
sums :: [Float]->[Float]
sums = sumsaux 0
  where
  sumsaux :: Float -> [Float] -> [Float]
  sumsaux s (x:xs) =
    newS : sumsaux newS xs  where newS = s+x
```

Τέλος, η `cosapprox` είναι απλά εφαρμογή της `sums` στο αποτέλεσμα της `cosdiffapprox`:

```
cosapprox :: Float->[Float]
cosapprox = sums.cosdiffapprox
```

5. Φτιάξτε μία συνάρτηση που να παίρνει έναν ακέραιο `n` και έναν πραγματικό `x` και να επιστρέφει τη `n`-οστή προσέγγιση του `cos x`, χρησιμοποιώντας τη συνάρτηση `cosapprox` του προηγούμενου ερωτήματος.

   §

```
cosapproxno :: Float->Int->Float
cosapproxno x n = (cosapprox x)!!n
```

6. Φτιάξτε μία συνάρτηση που να παίρνει δύο πραγματικούς αριθμούς `x` και `e` και επιστρέφει μία προσέγγιση του `cos x` με ακρίβεια `e`, χρησιμοποιώντας τη συνάρτηση `cosapprox` του προηγούμενου ερωτήματος.

   § Θεωρούμε ότι το `e` είναι θετικός. Για να έχουμε ακρίβεια `e`, θα πρέπει το απόλυτο της διαφοράς μεταξύ δύο διαδοχικών προσεγγίσεων να είναι μικρότερο του `e`. Η άπειρη λίστα με τις διαφορές δύο διαδοχικών προσεγγίσεων του `cos x` είναι η `cosdiffapprox x`. Επομένως, κοιτάμε σε ποιον δείκτη το απόλυτο ενός στοιχείου αυτής της λίστας γίνεται μικρότερο του `e`. Η λύση που δίνουμε είναι:

```
cosapproxwith :: Float->Float->Float
cosapproxwith x e =
      cosapprox x
  !! (length (takeWhile (>e) [abs d | d<-cosdiffapprox x]) + 1)
```

7. Γενικεύστε τη `dovetail3` ώστε να εφαρμόζει σε οποιονδήποτε αριθμό άπειρων λιστών.

   § Αφού πλέον το μέγεθος των παραγόμενων συνδυασμών δε μας είναι εκ των προτέρων γνωστό, θα τις παραστήσουμε με λίστες και όχι με πλειάδες. Επομένως η συνάρτησή μας θα είναι τύπου `[[Int]]->[[Int]]`.

   Αρχίζουμε αναπαριστώντας όλες τις λίστες φυσικών αριθμών κάποιου μεγέθους `len` των οποίων το άθροισμα είναι `sum`. Η συνάρτησή μας σε περίπτωση που το μέγεθος είναι `1` επιστρέφει μόνο μία λίστα, την προφανή:

```
alllists :: Int->Int->[[Int]]
alllists sum 1 = [[sum]]
```

Σε περίπτωση μεγαλύτερου μεγέθους, η `alllists` επιστρέφει όλες τις λίστες με κεφαλή `h` μεταξύ `0` και `sum` και ουρά τέτοια ώστε το συνολικό άθροισμα να είναι `sum`. Για να πάρουμε μία τέτοια ουρά, πρέπει να καλέσουμε αναδρομικά την `alllists` με άθροισμα `sum-h` και μέγεθος `len-1`:

```
alllists sum len =
  [h:t | h<-[0..sum], t<-alllists (sum-h) (len-1)]
```

Η `alllists` θα μας επιστρέφει όλες τις λίστες *δεικτών* των οποίων το άθροισμα είναι ένας συγκεκριμένος αριθμός. Αν έχουμε μια τέτοια λίστα δεικτών `ii`, καθώς και μία λίστα άπειρων λιστών `ll`, στην οποία θέλουμε να κάνουμε dovetailing, ο τρόπος να "εφαρμόσουμε" όλους τους δείκτες της `ii` στις αντίστοιχες λίστες της `ll` είναι να χρησιμοποιήσουμε τη `zipWith`:

```
zipWith (!!) ll ii
```

Επομένως, αν θέλουμε όλους τους συνδυασμούς στοιχείων λιστών `ll` από το βήμα `n` του dovetailing και μετά, ορίζουμε:

```
dovetailaux :: Int->[[Int]]->[[Int]]
dovetailaux n ll
 =    [zipWith (!!) ll ii | ii<-alllists n (length ll)]
   ++ dovetailaux (n+1) ll
```

δηλ. παίρνουμε τους δείκτες από την `alllists n (length ll)`, και καλούμε τη `zipWith (!!) ll` για να συσχετίσουμε τους δείκτες αυτούς με τις λίστες `ll`. Επίσης, καλούμε τη `dovetailaux` αναδρομικά για `n+1`, ώστε να παραχθούν και τα επόμενα βήματα του dovetailing.

Τέλος, η συνάρτησή μας `dovetail` ξεκινάει αυτή τη διαδικασία από το βήμα `0`:

```
dovetail :: [[Int]]->[[Int]]
dovetail = dovetailaux 0
```

8. Αποδείξτε ότι για κάθε `c` και `n<-[0..]` ισχύει

   ```
   ∀i<-[0..n-1].  (replicate n c)!!i == c
   ```

   Ο ορισμός του `replicate` είναι:

```
replicate 0 c = []
replicate n c = c : replicate(n-1)c
```

§ Απόδειξη με επαγωγή φυσικών στο `n`. Η βάση της επαγωγής είναι τετριμμένη, λόγω χρήσης καθολικού ποσοδείκτη πάνω σε κενό σύνολο.

Το βήμα της επαγωγής αποδεικνύεται ως εξής. Έστω

$$\forall \texttt{i<-[0..n-1].}\ \ \texttt{(replicate n c)!!i == c}$$

(επαγωγική υπόθεση) και `i<-[0..n]`. Θα πρέπει να αποδείξουμε:

$$\texttt{(replicate(n+1)c)!!i == c}$$

Παίρνουμε δύο περιπτώσεις. Στην πρώτη έχουμε `i==0` και η απόδειξη προχωράει ως εξής:

$$\texttt{(replicate(n+1)c)!!i} \ = \ \texttt{(c:replicate n c)!!0} \ = \ \texttt{c}$$

Στη δεύτερη έχουμε `i>0` και η απόδειξη είναι

```
    (replicate(n+1)c)!!i
=   (c:replicate n c)!!i
=   (replicate n c)!!(i-1)   επαγ. υπόθ. και i-1 <- [0..n-1]
=   c
```