ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΜΑΘΗΜΑ: ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ
ΑΚΑΔ. ΕΤΟΣ: 2024-25

1η Σειρά Προγραμματιστικών Ασκήσεων (Haskell)

Οι ασκήσεις αυτές είναι **ατομικές**. Οι απαντήσεις θα πρέπει να υποβληθούν με **turnin**, το αργότερο μέχρι την **Δευτέρα 28 Απριλίου 2025, ώρα 23:59**. Οι δύο σειρές προγραμματιστικών ασκήσεων δίνουν 1.5 μονάδες bonus στον τελικό βαθμό της τελικής και της επαναληπτικής εξέτασης, υπό την προϋπόθεση ότι ο βαθμός του αντίστοιχου γραπτού συν τον βαθμό της εξέτασης του εργαστηρίου είναι τουλάχιστον 5. Η βαθμολογία των προγραμματιστικών ασκήσεων δεν μεταφέρεται σε άλλη ακαδημαϊκή χρονιά και δεν λαμβάνεται υπόψη στη βαθμολογία της επί πτυχίω εξέτασης.

Πριν ξεκινήσετε να γράφετε τα προγράμματα που ζητούνται στις ασκήσεις της σειράς αυτής, **διαβάστε πολύ προσεκτικά τις αναλυτικές οδηγίες** που ακολουθούν.

Οδηγίες

- Για να εγκαταστήσετε τη Haskell στον υπολογιστή σας, μπορείτε να κατεβάσετε το διερμηνέα hugs από το σύνδεσμο

  https://www.haskell.org/hugs/pages/downloading-May2006.htm

  Συνοπτικές οδηγίες για τη χρήση του hugs υπάρχουν στις σημειώσεις του μαθήματος.

- Για τη συγγραφή των προγραμμάτων επιτρέπεται να χρησιμοποιήσετε προκαθορισμένες συναρτήσεις και προκαθορισμένους τελεστές **μόνο εφόσον αναφέρονται στις σημειώσεις του μαθήματος**. Δεν επιτρέπεται η χρήση του import.

- Για τη συγγραφή των συναρτήσεων θα πρέπει να χρησιμοποιήσετε το αρχείο πρότυπο Haskell.hs στο οποίο υπάρχουν έτοιμες οι δηλώσεις τύπων των συναρτήσεων που θα πρέπει να κατασκευάσετε καθώς και μια ισότητα που ορίζει τις συναρτήσεις ώστε να επιστρέφουν μια προκαθοριμένη τιμή για όλες τις τιμές των ορισμάτων. Για να απαντήσετε σε μια άσκηση θα πρέπει να αντικαταστήσετε την παραπάνω ισότητα με τις κατάλληλες ισότητες που ορίζουν την τιμή της συνάρτησης. **Δεν θα πρέπει να τροποποιήσετε το τύπο ούτε το όνομα της συνάρτησης**.

- Μπορείτε να χρησιμοποιήσετε όσες βοηθητικές συναρτήσεις θέλετε, οι οποίες θα καλούνται από τις συναρτήσεις που σας ζητείται να υλοποιήσετε. Σε καμία περίπτωση δεν θα πρέπει να προσθέσετε άλλα ορίσματα στις συναρτήσεις που σας ζητούνται (καθώς αυτό συνεπάγεται αλλαγή του τύπου τους).

- **Αν χρησιμοποιήσετε προκαθορισμένες συναρτήσεις ή τελεστές που δεν αναφέρονται στις σημειώσεις του μαθήματος ή αν χρησιμοποιήσετε το import για να ενσωματώσετε έτοιμο κώδικα, η αντίστοιχη άσκηση δεν θα βαθμολογηθεί.**

- Ο έλεγχος της ορθότητας των απαντήσεων θα γίνει με ημι-αυτόματο τρόπο. Σε καμία περίπτωση δεν θα πρέπει ο βαθμολογητής να χρειάζεται να κάνει παρεμβάσεις στο αρχείο που θα υποβάλετε. Συνεπώς θα πρέπει να λάβετε υπόψη τα παρακάτω:

  1. Κάθεμία από τις συναρτήσεις που σας ζητείται να υλοποιήσετε θα πρέπει να έχει το συγκεκριμένο όνομα και το συγκεκριμένο τύπο που περιγράφεται στην εκφώνηση της αντίστοιχης άσκησης και που υπάρχει στο αρχείο πρότυπο Haskell.hs. **Αν σε κάποια άσκηση το όνομα ή ο τύπος της συνάρτησης δεν συμφωνεί με αυτόν που δίνεται στην εκφώνηση, η άσκηση δεν θα βαθμολογηθεί**.
  2. Το αρχείο που θα παραδώσετε δεν θα πρέπει να περιέχει συντακτικά λάθη. Αν υπάρχουν τμήματα κώδικα που περιέχουν συντακτικά λάθη, τότε θα πρέπει να τα διορθώσετε ή να τα αφαιρέσετε πριν από την παράδοση. **Αν το αρχείο που θα υποβάλετε περιέχει συντακτικά λάθη, τότε οι ασκήσεις στις οποίες εμφανίζονται αυτά τα λάθη θα μηδενιστούν.**.
  3. Κατα τη διόρθωση των ασκήσεων οι βαθμολογητές δεν θα κάνουν κλήσεις στις βοηθητικές συναρτήσεις που ενδεχομένως θα χρησιμοποιήσετε. Η χρήση των βοηθητικών συναρτήσεων θα πρέπει να γίνεται μέσα από τις συναρτήσεις που σας ζητείται να υλοποιήσετε.

- Για υποβολή με turnin γράψτε:

turnin Haskell@myy401 Haskell.hs

Ασκηση 1.

Μία εταιρία κινητής τηλεφωνίας χρεώνει κάθε κλήση διάρκειας μέχρι 3 λεπτά προς οποιονδήποτε αριθμό με 0.58 ΕΥΡΩ και αν η διάρκεια της κλήσης υπερβεί τα 3 λεπτά, τότε ο επιπλέον χρόνος χρεώνεται με 0.003 ΕΥΡΩ το δευτερόλεπτο. Κλήσεις με μηδενική διάρκεια δεν χρεώνονται.

Γράψτε μία συνάρτηση `cost` σε Haskell η οποία θα δέχεται ως ορίσματα τις ώρες έναρξης και λήξης μίας κλήσης και θα υπολογίζει τη συνολική χρέωση για την κλήση. Η ώρα αναπαρίσταται ως μία τριάδα ακεραίων (για παράδειγμα η ώρα 15:18:31 αναπαρίσταται ως $(15, 18, 31)$). Ο τύπος της συνάρτησης θα πρέπει να είναι `(Int,Int,Int)->(Int,Int,Int)->Float`. Μπορείτε να υποθέσετε ότι τα δύο ορίσματα είναι πάντοτε έγκυρα (δηλαδή αντιστοιχούν σε σωστές ένδειξεις ώρας) και επιπλέον ότι η διάρκεια μίας κλήσης είναι μικρότερη από 24 ώρες.

Για τη μετατροπή ακέραιου σε πραγματικό χρησιμοποιήστε τη συνάρτηση `fromIntegral`.

Για έλεγχο χρησιμοποιήστε τις παρακάτω τιμές:

```
Main> cost (17,23,15) (17,23,15)
0.0
Main> cost (8,12,3) (8,12,58)
0.58
Main> cost (12,0,35) (12,1,24)
0.58
Main> cost (16,58,35) (17,0,0)
0.58
Main> cost (23,59,42) (0,1,40)
0.58
Main> cost (14,32,8) (14,35,17)
0.607
Main> cost (14,57,4) (15,0,23)
0.637
Main> cost (23,59,59) (0,2,59)
0.58
Main> cost (3,15,22) (11,55,8)
93.598
Main> cost (19,43,48) (1,5,7)
57.877
```

Ασκηση 2.

Μπορούμε να μετατρέψουμε έναν δεδομένο θετικό ακέραιο αριθμό σε έναν μονοψήφιο αριθμό με την παρακάτω διαδικασία: ενώσο ο αριθμός δεν είναι μονοψήφιος τον αντικαθιστούμε με το γινόμενο των μη μηδενικών ψηφίων του. Για παράδειγμα: 1978→504→20→2. Αν ο δεδομένος αριθμός είναι μονοψήφιος, τότε το αποτέλεσμα ισούται με τον ίδιο τον αριθμό.

Γράψτε μία συνάρτηση `compress` σε Haskell, η οποία θα δέχεται ως όρισμα έναν ακέραιο αριθμό n και θα τον μετατρέπει σε μονοψήφιο εφαρμόζοντας την παραπάνω διαδικασία. Ο τύπος της συνάρτησης θα πρέπει να είναι `Integer->Integer`. Μπορείτε να υποθέσετε ότι ο n είναι θετικός ακέραιος αριθμός.

Δεν επιτρέπεται να χρησιμοποιήσετε λίστες.

Για έλεγχο χρησιμοποιήστε τις παρακάτω τιμές:

```
Main> compress 7
7
Main> compress 32
6
Main> compress 35
5
Main> compress 39
4
Main> compress 58
4
Main> compress 7235
2
Main> compress 87251
3
Main> compress 25522525
1
Main> compress (11^15)
2
Main> compress (13^7128)
8
```

Ασκηση 3.

Είναι γνωστό από τα μαθηματικά ότι αν a, k, m είναι μη αρνητικοί ακέραιοι αριθμοί με $m \geq 2$ τότε το σύνολο $\{n \in \mathbb{N} \setminus \{0\} \mid (n+a)^k < m^n\}$ περιέχει άπειρα στοιχεία. Γράψτε μία συνάρτηση search σε Haskell η οποία θα δέχεται ως ορίσματα τρεις ακέραιους αριθμούς a, k, m και θα επιστρέφει τον ελάχιστο θετικό ακέραιο n για τον οποίο ισχύει $(n+a)^k < m^n$ (δηλαδή το ελάχιστο στοιχείο του συνόλου που περιγράφεται παραπάνω). Ο τύπος της συνάρτησης θα πρέπει να είναι `Integer->Integer->Integer->Integer`. Μπορείτε να υποθέσετε ότι a, k, m είναι μη αρνητικοί ακέραιοι αριθμοί και ότι $m \geq 2$.

Δεν επιτρέπεται να χρησιμοποιήσετε λίστες.

Για έλεγχο χρησιμοποιήστε τις παρακάτω τιμές:

```
Main> search 0 2 2
1
Main> search 1 2 2
6
Main> search 2 2 2
7
Main> search 2 5 2
24
Main> search 5 2 2
8
Main> search 1 10 3
32
Main> search 1 100 2
997
Main> search 1000 2 3
13
Main> search 100 100 100
117
Main> search 1000 1000 1000
1108
```

Ασκηση 4.

Γράψτε μία συνάρτηση sum2025 σε Haskell η οποία θα δέχεται ως ορίσματα δύο μη αρνητικούς άκεραιους αριθμούς m, n και θα επιστρέφει το άθροισμα:

$$\sum_{i=m}^{n}(n+i)^m$$

Ο τύπος της συνάρτησης θα πρέπει να είναι `Integer->Integer->Integer`. Μπορείτε να υποθέσετε ότι m, n είναι μη αρνητικοί ακέραιοι αριθμοί.

Δεν επιτρέπεται να χρησιμοποιήσετε λίστες.

Για έλεγχο χρησιμοποιήστε τις παρακάτω τιμές:

```
Main> sum2025 1 4
26
Main> sum2025 2 4
149
Main> sum2025 3 4
855
Main> sum2025 4 4
4096
Main> sum2025 2 12
4081
Main> sum2025 5 15
124854125
Main> sum2025 20 20
109951162777600000000000000000000
Main> sum2025 10 100
191097881558651991974732 5
```

Ασκηση 5.

Τα αποτελέσματα των αγώνων μίας ποδοσφαιρικής ομάδας κατά τη διάρκεια του πρωτάθληματος, μπορούν να αναπαρασταθούν ως μία λίστα από ζεύγη ακεραίων. Το κάθε ζεύγος αναπαριστά το αποτέλεσμα ενός αγώνα, όπου το πρώτο στοιχείο του ζεύγους αντιστοιχεί στα τέρματα τα οποία πέτυχε η ομάδα και το δεύτερο στοιχείο στα τέρματα που δέχτηκε από την αντίπαλη ομάδα στον συγκεκριμένο αγώνα.

Με δεδομένη μία λίστα που περιλαμβάνει τα αποτέλεσματα μίας ομάδας στο πρωτάθλημα, θέλουμε να κατασκευάσουμε μία πεντάδα αριθμών που να περιγράφει τα παρακάτω στατιστικά της ομάδας:

- το συνολικό πλήθος αγώνων που έχει παίξει
- τους συνολικούς βαθμούς που έχει κερδίσει, με δεδομένο ότι για κάθε νίκη κερδίζει τρεις βαθμούς και για κάθε ισοπαλία έναν βαθμό
- το συνολικό πλήθος τερμάτων που έχει πετύχει
- το συνολικό πλήθος τερμάτων που έχει δεχτεί
- την διαφορά τερμάτων στο καλύτερο αποτέλεσμα που έχει φέρει. Η διαφορά αυτή θα είναι θετική αν η ομάδα έχει επιτύχει τουλάχιστον μία νίκη, μηδέν αν δεν έχει επιτύχει νίκη αλλά έχει τουλάχιστον μία ισοπαλία και αρνητική αν έχει ηττηθεί σε όλους τους αγώνες.

Γράψτε μία συνάρτηση `statistics` σε Haskell, η οποία θα δέχεται ως όρισμα τη λίστα με τα αποτελέσματα μίας ομάδας και θα επιστρέφει μία πεντάδα ακεραίων με τα στατιστικά στοιχεία που περιγράφονται παραπάνω. Αν η λίστα είναι κενή, τότε θα πρέπει να επιστρέφεται η πεντάδα (0,0,0,0,0). Ο τύπος της συνάρτησης θα πρέπει να είναι `[(Int,Int)]->(Int,Int,Int,Int,Int)`. Μπορείτε να υποθέσετε ότι η λίστα είναι πεπερασμένη και ότι κάθε ζεύγος στη λίστα απαρτίζεται από δύο μη αρνητικούς αριθμούς (δηλαδή η λίστα περιέχει μόνο έγκυρα αποτελέσματα). Δεν υπάρχει περιορισμός για το πλήθος τερμάτων που μπορεί να επιτύχει μία ομάδα σε έναν αγώνα, ούτε για το μήκος της λίστας.

Για έλεγχο χρησιμοποιήστε τις παρακάτω τιμές:

```
Main> statistics []
(0,0,0,0,0)
Main> statistics [(1,5)]
(1,0,1,5,-4)
Main> statistics [(0,1),(1,3),(1,2)]
(3,0,2,6,-1)
Main>statistics [(3,1),(5,2),(7,0)]
(3,9,15,3,7)
Main> statistics [(1,1),(2,2),(4,4)]
(3,3,7,7,0)
Main> statistics [(8,1),(2,3),(3,3),(2,0)]
(4,7,15,7,7)
Main> statistics [(0,4),(2,2),(2,3),(0,0)]
(4,2,4,9,0)
Main> statistics [(1,1),(3,0),(0,2),(4,3),(7,1),(3,3),(1,4),(2,1)]
(8,14,21,15,6)
```

Ασκηση 6.

Θέλουμε να σχηματίσουμε μία λίστα με όλες τις λέξεις οι οποίες περιέχονται μέσα σε μία δεδομένη συμβολοσειρά. Ονομάζουμε λέξη ένα μη κενό τμήμα της συμβολοσειράς, το οποίο αποτελείται από συνεχόμενα σύμβολα που είναι όλα λατινικά γράμματα (κεφαλαία ή μικρά) και το οποίο δεν περιέχεται σε ένα ευρύτερο τμήμα με την ίδια ιδιότητα (με άλλα λόγια αριστερά και δεξία μίας λέξης που περιέχεται σε μία συμβολοσειρά δεν βρίσκεται γράμμα του λατινικού αλφαβήτου). Για παράδειγμα στη συμβολοσειρά `"Rockabilly Boogie"` περιέχονται οι λέξεις `"Rockabilly"` και `"Boogie"` , ενώ οι συμβολοσειρές `"Rock"` και `"billy"` δεν αποτελούν λέξεις.

Γράψτε μία συνάρτηση `wordList` σε Haskell, η οποία θα δέχεται ως όρισμα μία συμβολοσειρά και θα επιστρέφει μία λίστα με όλες τις λέξεις που περιέχονται σε αυτή, με τη σειρά εμφάνισής τους. Ο τύπος της συνάρτησης θα πρέπει να είναι `String->[String]`.

Για έλεγχο χρησιμοποιήστε τις παρακάτω τιμές:

```
Main> wordList ""
[]
Main> wordList "HAZEL"
["HAZEL"]
Main> wordList "Three Simple Words"
["Three","Simple","Words"]
Main> wordList "s.i.n.g.l.e.s"
["s","i","n","g","l","e","s"]
Main> wordList "BEGIN  ...  END"
["BEGIN","END"]
Main> wordList "Would you like some pizza?"
["Would","you","like","some","pizza"]
Main> wordList "* is a star"
["is","a","star"]
Main> wordList "477392-0900024-4234324324"
[]
Main> wordList "------{middle}------"
["middle"]
Main> wordList "[USER-ID:12Ag$Z?5h-65S], E-mail: iam4got10@cs.uoi.gr"
["USER","ID","Ag","Z","h","S","E","mail","iam","got","cs","uoi","gr"]
```

Ασκηση 7.

Μπορούμε να αναπαραστήσουμε πίνακες, χρησιμοποιώντας λίστες τα στοιχεία των οποίων είναι λίστες. Κάθε εσωτερική λίστα περιέχει τα στοιχεία μίας γραμμής του πίνακα. Για να αποτελεί μία λίστα από λίστες αναπαράσταση ενός πίνακα θα πρέπει να είναι μη κενή και τα στοιχεία της να είναι μη κενές λίστες που όλες έχουν το ίδιο πλήθος στοιχείων.

(α) Γράψτε μία συνάρτηση `size` σε Haskell, η οποία θα δέχεται ως είσοδο μία λίστα από λίστες οποιουδήποτε τύπου και αν η λίστα αναπαριστά έναν πίνακα με m γραμμές και n στήλες θα επιστρέφει το ζεύγος (m, n), αλλιώς θα επιστρέφει το $(0, 0)$. Ο τύπος της συνάρτησης θα πρέπει να είναι `[[u]]->(Int,Int)`.

(β) Γράψτε μία συνάρτηση `transpose` σε Haskell, η οποία θα δέχεται ως είσοδο μία λίστα από λίστες οποιουδήποτε τύπου και αν η λίστα αναπαριστά έναν πίνακα A θα επιστρέφει τη λίστα που αναπαριστά τον ανάστροφο του A, αλλιώς θα επιστρέφει την κενή λίστα. Ο τύπος της συνάρτησης θα πρέπει να είναι `[[u]]->[[u]]`.

Υπενθυμίζεται ότι ο ανάστροφος του πίνακα A είναι ο πίνακας A^T, για τον οποίο ισχύει $(A^T)_{i,j} = A_{j,i}$ (δηλαδή το στοιχείο στην i-οστή γραμμή και στη j-οστή στήλη του ανάστροφου πίνακα A^T είναι το στοιχείο που βρίσκεται στη j-οστή γραμμή και στην i-οστή στήλη του A).

(γ) Γράψτε μία συνάρτηση `matrixmult` σε Haskell, η οποία θα δέχεται ως είσοδο δύο λίστες από λίστες ακεραίων και αν οι λίστες αναπαριστούν δύο πίνακες A και B και το πλήθος των στηλών του A είναι ίσο με το πλήθος των γραμμών του B θα επιστρέφει τη λίστα από λίστες που αναπαριστά το γινόμενο των A και B. Ο τύπος της συνάρτησης θα πρέπει να είναι `[[Int]]->[[Int]]->[[Int]]`.

Υπενθυμίζεται ότι το γινόμενο των πινάκων A και B είναι ο πίνακας C, για τον οποίο ισχύει $C_{i,j} = \sum_{k=1}^{n} A_{i,k} \cdot B_{k,j}$ (όπου n είναι το πλήθος των στηλών του A και το πλήθος των γραμμών του B).

Για έλεγχο χρησιμοποιήστε τις παρακάτω τιμές:

```
Main> size []
(0,0)
Main> size [[1,3,2,8],[5,6,7,8],[4,2,0,0]]
(3,4)
Main> size [[],[],[],[]]
(0,0)
Main> size [[3,2],[3,4,5],[5,6]]
(0,0)
Main> size ["abcde","01245"]
(2,5)
Main> size [[1,2,3,4,5]]
(1,5)
```

```
Main> size [[1],[2],[3],[4],[5]]
(5,1)
Main> transpose []
[]
Main> transpose [[1,3,2,8],[5,6,7,8],[4,2,0,0]]
[[1,5,4],[3,6,2],[2,7,0],[8,8,0]]
Main> transpose [[3,2],[3,4,5],[5,6]]
[]
Main> transpose ["abcde","01245"]
["a0","b1","c2","d4","e5"]
Main> transpose [[1,2,3,4,5]]
[[1],[2],[3],[4],[5]]
Main> transpose [[1],[2],[3],[4],[5]]
[[1,2,3,4,5]]
Main> matrixmult [[1,2,3,4,5]] [[1],[2],[3],[4],[5]]
[[55]]
Main> matrixmult [[1],[2],[3],[4],[5]] [[1,2,3,4,5]]
[[1,2,3,4,5],[2,4,6,8,10],[3,6,9,12,15],[4,8,12,16,20],[5,10,15,20,25]]
Main> matrixmult [[2,3,4,5],[5,7,8,3],[6,4,2,7]] [[3,5],[6,8],[2,3],[9,7]]
[[77,81],[100,126],[109,117]]
```

Ασκηση 8.

Ένα ρομπότ μπορεί να κινείται σε ένα επίπεδο, το οποίο είναι χωρισμένο σε τετράγωνους τομείς. Κάθε τομέας περιγράφεται από ένα ζεύγος ακεραίων αριθμών που καθορίζει τις συντεταγμένες του. Το ρομπότ για να κινηθεί από έναν τομέα αφετηρία (x_1, y_1) σε έναν τομέα προορισμό (x_2, y_2), αρχικά κινείται διαγώνια προς την κατάλληλη κατεύθυνση, μέχρι κάποια από τις δύο συνιστώσες του να γίνει ίση με την αντίστοιχη του τομέα προορισμού. Στη συνέχεια, κινείται οριζόντια ή κάθετα μέχρι να φτάσει στον τομέα προορισμό. Για πάραδειγμα, για να κινηθεί από τον τομέα $(-2, 3)$ στον τομέα $(4, 5)$, το ρομπότ θα περάσει από τους τομείς $(-1, 4)$, $(0, 5)$, $(1, 5)$, $(2, 5)$ και $(3, 5)$, ενώ για να κινηθεί από τον τομέα $(3, 8)$ στον τομέα $(9, 0)$, θα περάσει από τους τομείς $(4, 7)$, $(5, 6)$, $(6, 5)$, $(7, 4)$, $(8, 3)$, $(9, 2)$ και $(9, 1)$.

Το ρομπότ καθημερινά εκτελεί μία σειρά από εργασίες που πρέπει να γίνουν με προκαθορισμένη σειρά σε κάποιους τομείς $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. Το ρομπότ ξεκινώντας από την αρχική του θέση που είναι ο τομέας $(0, 0)$, θα πρέπει να μεταβεί πρώτα στον τομέα (x_1, y_1), να εκτέλέσει εκεί την πρώτη εργασία, στη συνέχεια να μεταβεί στον τομέα (x_2, y_2) και να εκτέλέσει εκεί τη δεύτερη εργασία και να συνεχίσει με τον ίδιο τρόπο, μέχρι να ολοκληρώσει και την τελευταία εργασία του. Τέλος από τον τομέα (x_n, y_n) πρέπει να επιστρέψει στην αρχική του θέση $(0, 0)$.

Γράψτε μία συνάρτηση `trace` σε Haskell, η οποία θα δέχεται ως όρισμα μία λίστα από ζεύγη ακεραίων, η οποία περιέχει τις συντεταγμένες των τομέων στις οποίες θα πρέπει να εκτελεστούν οι εργασίες από το ρομπότ και θα επιτρέφει μία λίστα του ίδιου τύπου, η οποία θα περιέχει όλους τους τομείς που θα επισκεφτεί το ρομπότ ξεκινώντας από τη θέση $(0, 0)$ έτσι ώστε να εκτέλεσει τις απαραίτητες εργασίες σε όλους τους τομείς της λίστας και να επιστρέψει στη θέση $(0, 0)$. Μπορείτε να υποθέσετε ότι η λίστα δεν περιέχει το στοιχείο $(0, 0)$ και ότι διαδοχικές εργασίες εκτελούνται σε διαφορετικούς τομείς. Ο τύπος της συνάρτησης θα πρέπει να είναι `[(Int,Int)]->[(Int,Int)]`

Για έλεγχο χρησιμοποιήστε τις παρακάτω τιμές:

```
Main> trace [(3,8)]
[(0,0),(1,1),(2,2),(3,3),(3,4),(3,5),(3,6),(3,7),(3,8),(2,7),(1,6),(0,5),(0,4),(0,3),
(0,2),(0,1),(0,0)]
Main> trace [(5,5),(-5,5)]
[(0,0),(1,1),(2,2),(3,3),(4,4),(5,5),(4,5),(3,5),(2,5),(1,5),(0,5),(-1,5),(-2,5),
(-3,5),(-4,5),(-5,5),(-4,4),(-3,3),(-2,2),(-1,1),(0,0)]
Main> trace [(5,3),(-2,-6),(-1,1),(3,-7)]
[(0,0),(1,1),(2,2),(3,3),(4,3),(5,3),(4,2),(3,1),(2,0),(1,-1),(0,-2),(-1,-3),(-2,-4),
(-2,-5),(-2,-6),(-1,-5),(-1,-4),(-1,-3),(-1,-2),(-1,-1),(-1,0),(-1,1),(0,0),(1,-1),
(2,-2),(3,-3),(3,-4),(3,-5),(3,-6),(3,-7),(2,-6),(1,-5),(0,-4),(0,-3),(0,-2),(0,-1),
0,0)]
Main> trace [(i,2*(-1)^i) | i <- [1..12]]
[(0,0),(1,-1),(1,-2),(2,-1),(2,0),(2,1),(2,2),(3,1),(3,0),(3,-1),(3,-2),(4,-1),(4,0),
(4,1),(4,2),(5,1),(5,0),(5,-1),(5,-2),(6,-1),(6,0),(6,1),(6,2),(7,1),(7,0),(7,-1),
(7,-2),(8,-1),(8,0),(8,1),(8,2),(9,1),(9,0),(9,-1),(9,-2),(10,-1),(10,0),(10,1),(10,2)
,(11,1),(11,0),(11,-1),(11,-2),(12,-1),(12,0),(12,1),(12,2),(11,1),(10,0),(9,0),(8,0),
(7,0),(6,0),(5,0),(4,0),(3,0),(2,0),(1,0),(0,0)]
```

Ασκηση 9.

Γράψτε μία συνάρτηση `generating` υψηλότερης τάξης σε Haskell, η οποία θα δέχεται ως ορίσματα

(α) μία συνάρτηση f από ακέραιους σε πραγματικούς και

(β) έναν μη αρνητικό ακέραιο αριθμό k

και θα επιστρέφει ως αποτέλεσμα τη συνάρτηση g_k από πραγματικούς σε πραγματικούς, η οποία ορίζεται με τον παρακάτω τρόπο:

$$g_k(z) = \sum_{i=0}^{k} f(i) \cdot z^i$$

Ο τύπος της συνάρτησης `generating` θα πρέπει να είναι `(Int->Double)->Int->(Double->Double)`.

Για έλεγχο χρησιμοποιήστε τις παρακάτω τιμές:

```
Main> generating (\n->2.0^n) 5 0.25
1.96875
Main> generating (\n->2.0^n) 1000 0.25
2.0
Main> generating (\n->fromIntegral n) 1000 0.2
0.3125
Main> generating (\n->fromIntegral n) 1000 0.6
3.75
Main> generating (\n->fromIntegral n^2) 1000 0.8
180.0
```

Ασκηση 10.

Γράψτε μία συνάρτηση apply υψηλότερης τάξης σε Haskell, η οποία θα δέχεται ως ορίσματα

(α) μία λίστα συναρτήσεων με πεδίο ορισμού οποιονδήποτε τύπο και πεδίο τιμών οποιονδήποτε διατεταγμένο τύπο και

(β) μία λίστα με στοιχεία του ίδιου τύπου με το πεδίο ορισμού των συναρτήσεων της λίστας του (α)

και θα επιστρέφει ως αποτέλεσμα μία λίστα η οποία θα περιέχει όλες τις τιμές που προκύπτουν εφαρμόζοντας κάθε συνάρτηση της πρώτης λίστας σε κάθε στοιχείο της δεύτερης λίστας. Η επιστρεφόμενη λίστα θα πρέπει να είναι ταξινομημένη σε αύξουσα τάξη, χωρίς επαναλήψεις στοιχείων. Ο τύπος της συνάρτησης apply θα πρέπει να είναι `Ord u => [v->u]->[v]->[u]`.

Για έλεγχο χρησιμοποιήστε τις παρακάτω τιμές:

```
Main> apply [abs] [-1]
[1]
Main> apply [(^2)] [3,2,7,8,4]
[4,9,16,49,64]
Main> apply [(2^),(^2),(^3),(^4)] [10]
[100,1000,1024,10000]
Main> apply [(^0),(0^),(\x->div x x),(\x->mod x x)] [1..1000]
[0,1]
Main> apply [(2^),(^2),(^3),(^4)] [2..8]
[4,8,9,16,25,27,32,36,49,64,81,125,128,216,256,343,512,625,1296,2401,4096]
Main> apply [head,last] ["abc","aaaa","cbbc","cbbca"]
"ac"
Main> apply [head.tail,last.init] ["abc","aaaa","cbbc","cbbca"]
"abc"
Main> apply [reverse,(++"ing"),reverse.(++"ing"),(++"ing").reverse] ["play","do"]
["doing","gniod","gniyalp","od","oding","playing","yalp","yalping"]
Main> apply [\x->mod x 10, \x->rem x 10] [-100..100]
[-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9]
Main> apply [(*5)] (apply [(`div`5)] [1..100])
[0,5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100]
```