Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Επώνυμο:
Όνομα:
Αριθμός Μητρώου:

1	0.5	
2	1.5	
3	1.5	
4	1	
5	1	
6	1	
7	1.5	
Σ	8	

ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ Ι ***5/2/2021***

Τελικό Διαγώνισμα (Εξέταση "επί πτυχίω")

Το διαγώνισμα αυτό έχει ερωτήσεις 8 βαθμών συνολικά.
Για τον υπολογισμό του τελικού σας βαθμού θα προστεθούν και οι βαθμοί των ασκήσεών σας την πιο πρόσφατη χρονιά που δηλώσατε το μάθημα και παραδώσατε τουλάχιστον μία σειρά ασκήσεων.

Γράψτε τις απαντήσεις σας σε ένα αρχείο κειμένου με όνομα el17042.txt (προφανώς θα χρησιμοποιήστε τον πραγματικό αριθμό μητρώου σας) και υποβάλετέ τα στο moodle. Στην πρώτη γραμμή του αρχείου γράψτε οπωσδήποτε το ονοματεπώνυμο και τον αριθμό μητρώου σας. Αν χρειαστεί να υποβάλετε σχήματα ή οτιδήποτε δεν μπορεί να γραφεί σε αρχείο κειμένου, μπορείτε να υποβάλετε ένα zip που να περιέχει όλες τις απαντήσεις σας.

*Η **διάρκεια της εξέτασης** είναι 1:30 ώρα και μετά το τέλος της θα έχετε άλλα 30 λεπτά για να υποβάλετε τις απαντήσεις σας. Χρησιμοποιήστε το χρόνο όπως νομίζετε καλύτερα αλλά απαντήσεις που θα υποβληθούν μετά το πέρας των δύο ωρών δε θα γίνουν δεκτές.*

Αν στις απαντήσεις σας «δανειστείτε» οτιδήποτε από ευρέως διαθέσιμη πηγή, φροντίστε να το αναφέρετε ρητά και να παραπέμπετε στην πηγή σας. Οποιεσδήποτε άλλες αδικαιολόγητες «ομοιότητες» μεταξύ γραπτών θα θεωρούνται αντιγραφή και τα αντίστοιχα θέματα θα μηδενίζονται.

1. Γραμματικές (0.5 βαθμός)

Δίνεται η παρακάτω γραμματική για τις ισοσκελισμένες ακολουθίες παρενθέσεων:

`<S> ::= <S> <S> | ( <S> ) | ()`

Να δείξετε ότι η γραμματική είναι διφορούμενη (ambiguous).

2. Προγραμματισμός σε ML (0.5 + 0.5 + 0.5 = 1.5 βαθμοί)

α) Να ορίσετε τον ορισμό της συνάρτησης **`reduceB`** η οποία παίρνει ως ορίσματα μια τιμή **`g`** κάποιου τύπου **`'b`**, μια συνάρτηση **`F`** τύπου **`'a * 'b -> 'b`**, και μια λίστα **$[a_1,\ldots,a_{n-1},a_n]$** από στοιχεία τύπου **`'a`** και επιστρέφει την τιμή **$F(a_1,\ldots,F(a_{n-1},F(a_n,g))\ldots)$**.

Χρησιμοποιήστε τη συνάρτηση **`reduceB`** (ακόμα και αν δε λύσατε το παραπάνω υποερώτημα) για:

β) Να δώσετε έναν εναλλακτικό ορισμό της συνάρτησης **`len`** που επιστρέφει το μήκος μιας λίστας.

γ) Να ορίσετε τη συνάρτηση **`suffixes`** που επιστρέφει τη λίστα με τα επιθέματα (suffixes) μιας λίστας. Για παράδειγμα, για **`[1,2,3]`** η συνάρτηση επιστρέφει **`[[1,2,3], [2,3], [3], []`**.

Προσοχή! Για τα υποερωτήματα (β) και (γ) είναι υποχρεωτικό να ορίσετε τις ζητούμενες συναρτήσεις χρησιμοποιώντας τη συνάρτηση **`reduceB`**. Διαφορετικά δε θα βαθμολογηθούν.

3. Προγραμματισμός σε Python (0.3 + 1 + 0.2 = 1.5 βαθμοί)

Θα θεωρούμε ότι δύο λίστες ακεραίων αριθμών **A** και **B** είναι ίσες αν έχουν ακριβώς τα ίδια στοιχεία ανεξαρτήτως σειράς, δηλαδή αν η μία είναι αντιμετάθεση των στοιχείων της άλλης. Για παράδειγμα, οι λίστες [1, 2, 3] και [2, 3, 1] είναι ίσες, αλλά καμία από αυτές δεν είναι ίση με τη λίστα [1, 4, 2], ούτε με τη λίστα [1, 3, 1, 2].

α) Να γραφεί σε Python μία *κομψή και αποδοτική* συνάρτηση **`equal_list(A,B)`** που να ελέγχει αν οι δοθείσες λίστες είναι ίσες ή όχι.

Αυτή η σχέση ισότητας είναι σχέση ισοδυναμίας (equivalence relation), δηλαδή είναι σχέση ανακλαστική, μεταβατική και συμμετρική. Με αυτήν μπορούμε να διαμερίσουμε το σύνολο των λιστών ακεραίων αριθμών σε κλάσεις ισοδυναμίας (equivalence classes), δηλαδή σε μέγιστα δυνατά υποσύνολα που τα στοιχεία καθενός από αυτά να είναι λίστες ίσες μεταξύ τους.

β) Δίνεται μία λίστα **L**, τα στοιχεία της οποίας είναι λίστες ακεραίων αριθμών. Να γραφεί σε Python μία *κομψή και αποδοτική* συνάρτηση **`count_classes(L)`** που να βρίσκει σε πόσες διαφορετικές κλάσεις ισοδυναμίας διαμερίζονται τα στοιχεία της **L**.

Για παράδειγμα, **`count_classes([1,2,3], [1,4,2], [2,3,1]) == 2`**, γιατί η πρώτη και η τελευταία λίστα ανήκουν στην ίδια κλάση ισοδυναμίας, ενώ η δεύτερη σε άλλη.

γ) Να αναφερθεί η *χρονική* και η *χωρική* πολυπλοκότητα της υλοποίησης που δώσατε ως απάντηση στα υποερωτήματα (α) και (β).

4. Εγγραφές δραστηριοποίησης (1 βαθμός)

Γράψτε τη *συνοπτικότερη* συνάρτηση ML που μπορείτε να σκεφτείτε (όχι αυτή στις διαφάνειες!) η οποία δε θα λειτουργούσε σωστά αν το σύστημα υλοποίησης της γλώσσας ML χρησιμοποιούσε εγγραφές δραστηριοποίησης οργανωμένες σε στοίβα αλλά χωρίς συνδέσμους φωλιάσματος (nesting links). Εξηγήστε γιατί η συνάρτηση θα αποτύγχανε.

5. Ονόματα και δεσίματά τους (0.5 + 0.5 = 1 βαθμός)

Στο βιβλίο του Ken Arnold και James Gosling για τη γλώσσα προγραμματισμού Java υπάρχει το παρακάτω παράδειγμα για τους χώρους ονομάτων της γλώσσας και τη δυνατότητα χρησιμοποίησης του ίδιου ονόματος για διαφορετικούς σκοπούς:

```
class ReUse {
   ReUse ReUse(ReUse ReUse) {
      ReUse:
        for (;;) {
            if (ReUse.ReUse(ReUse) == ReUse)
                  break ReUse;
        }
        return ReUse;
   }
}
```

Δώστε απαντήσεις στις παρακάτω ερωτήσεις, πιθανώς αντιγράφοντας το πρόγραμμα και σημειώνοντας πάνω του:

α) Για κάθε εμφάνιση του ονόματος **`ReUse`** η οποία αποτελεί ορισμό, περιγράψτε τι είδους δέσιμο καθορίζει ο ορισμός αυτός.

β) Για κάθε εμφάνιση του ονόματος **`ReUse`** η οποία δεν αποτελεί ορισμό, βρείτε τον ορισμό που καθορίζει το δέσιμό του.

6. Πέρασμα παραμέτρων (1 βαθμός)

Γράψτε ένα παράδειγμα προγράμματος σε ML που να αποδεικνύει ότι η ML δε χρησιμοποιεί κλήση κατ' όνομα. Αναφέρετε τα αποτελέσματα που προκύπτουν στο παράδειγμά σας και εξηγήστε ποια αποτελέσματα θα προέκυπταν στην περίπτωση που η ML χρησιμοποιούσε κλήση κατ' όνομα.

7. Προγραμματισμός σε Prolog (0.3 + 0.4 + 0.3 = 1 βαθμός)

Εκτός από το πρόβλημα με τις βασίλισσες που είδαμε στις διαλέξεις, ένα άλλο γνωστό πρόβλημα σχετικό με το σκάκι είναι το πρόβλημα της *βόλτας του αλόγου* (*knight's tour*). Η πιο συνηθισμένη παραλλαγή του προβλήματος ζητάει να βρεθεί μια *κλειστή* βόλτα (δηλ. μια βόλτα που ξεκινάει και τελειώνει στο ίδιο τετράγωνο και επισκέπτεται όλα τα υπόλοιπα τετράγωνα της σκακιέρας ακριβώς μια φορά) αλλά στην άσκηση αυτή θα μας απασχολήσουν κάποια υποπροβλήματα.

(Αν θέλετε να θυμηθείτε πώς κινούνται τα άλογα στο σκάκι, δείτε εδώ.)

Έστω ότι η σκακιέρα έχει διαστάσεις **`N×N`**. Στην Prolog τα τετράγωνα μπορούν να αναπαρασταθούν με όρους της μορφής **`X/Y`**, όπου τα **`X`** και **`Y`** είναι ακέραιοι μεταξύ **`1`** και **`N`**.

α) Να ορίσετε το κατηγόρημα **`jump(N,Square1,Square2)`** το οποίο προσομοιώνει την κίνηση του αλόγου σε σκακιέρα διαστάσεων **`N×N`** ξεκινώντας από το τετράγωνο **`Square1`** και καταλήγοντας (σε ένα βήμα) στο τετράγωνο **`Square2`**. Μπορείτε να υποθέσετε ότι το τετράγωνο **`Square1`** θα είναι πάντα πλήρως γνωστό (instantiated) ενώ το τετράγωνο **`Square2`** μπορεί και να μην είναι. Δύο παραδείγματα:

```
?- jump(8, 1/1, S).
S = 3/2 ;
S = 2/3 ;
no
```

```
?- jump(8, 4/4, 5/Y).
Y = 6 ;
Y = 2 ;
no
```

β) Να ορίσετε το κατηγόρημα **`knightpath(N,Path)`** το οποίο επιτυγχάνει αν το όρισμα **`Path`** είναι μια λίστα από τετράγωνα που σχηματίζουν ένα επιτρεπτό μονοπάτι του αλόγου σε μια σκακιέρα διαστάσεων **`N×N`**. Τα επιτρεπτά μονοπάτια δεν είναι απαραίτητο να είναι κλειστές βόλτες (δηλ. βόλτες που έχουν την ίδια αρχή και τέλος), αλλά, με εξαίρεση το αρχικό τετράγωνο, δεν επιτρέπεται να επισκέπτονται κάποιο τετράγωνο πάνω από μια φορά.

γ) Με χρήση του κατηγορήματος **`knightpath/2`** να γράψετε μια ερώτηση που θα μπορούσαμε να κάνουμε σε ένα σύστημα Prolog αν θέλαμε να βρούμε ένα μονοπάτι με τέσσερις κινήσεις από το τετράγωνο **`2/1`** στη απέναντι πλευρά μιας 8x8 σκακιέρας (**`Y=8`**) το οποίο περνάει από το τετράγωνο **`5/4`** κατά ή μετά τη δεύτερη κίνηση.

Καλή επιτυχία!