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

Επώνυμο:
Όνομα:
Αριθμός Μητρώου:

1	0.75	
2	1.5	
3	1	
4	1.5	
5	1.75	
6	1.5	
Σ	8	

ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ Ι *Εαρινό 2020*

Τελικό Διαγώνισμα (Κανονική Εξέταση) #3

Το διαγώνισμα αυτό έχει ερωτήσεις 8 βαθμών συνολικά.

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 έστω 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.

Προσοχή! *Κάποια ερωτήματα εξαρτώνται από τον αριθμό μητρώου σας. Η απάντηση που θα δώσετε θα θεωρηθεί σωστή μόνο αν έχετε αντικαταστήσει σωστά τις σταθερές **AM_1**, **AM_2** και **AM_3** με τα τρία τελευταία δεκαδικά ψηφία του αριθμού μητρώου σας. Π.χ. αν ο αριθμός μητρώου σας είναι el17042, πρέπει να θεωρήσετε ότι σε αυτά τα ερωτήματα θα είναι **AM_1=0, AM_2=4** και **AM_3=2**.*

1. Γραμματικές (3 * 0.25 = 0.75 βαθμοί)

Έστω η παρακάτω γραμματική για αριθμητικές εκφράσεις:

```
<expr> ::= <expr> + <mult> | <mult>
<mult> ::= <mult> * <fact> | <fact>
<fact> ::= ( <expr> ) | a | b | c
```

Τροποποιήστε την παραπάνω γραμματική με τους εξής τρόπους (διαδοχικά):

α) Προσθέστε τελεστές αφαίρεσης και διαίρεσης (**-** και **/**) με τις συνηθισμένες προτεραιότητες και τη συνηθισμένη προσεταιριστικότητα.

β) Στη συνέχεια, προσθέστε και έναν αριστερά προσεταιριστικό τελεστή **%** με προτεραιότητα μεταξύ του **+** και του *****.

γ) Τέλος, προσθέστε και ένα δεξιά προσεταιριστικό τελεστή **=** με προτεραιότητα χαμηλότερη από όλους τους άλλους τελεστές.

2. Ερωτήσεις κατανόησης (6 * 0.25 = 1.5 βαθμοί)

Απαντήστε χωρίς αιτιολόγηση.

α) Έστω η διπλανή συνάρτηση σε Python. Κανονικοποιεί ένα διάνυσμα a, διαιρώντας κάθε στοιχείο με το συνολικό άθροισμα. Π.χ.

```
f([0, 1, 2, 3, 4])
= [0.0, 0.1, 0.2, 0.3, 0.4]

f([0, 1, 4, 9])
= [0.0, 0.018…, 0.072…, 0.163…, 0.29…]
```

```
def f(a):
  s = sum(a)
  return [x / s for x in a]
```

Μπορείτε να την καλέσετε και με αντικείμενα που δεν είναι λίστες, αρκεί να είναι “iterable”, π.χ.

```
f(range(5))
= [0.0, 0.1, 0.2, 0.3, 0.4]
```

Όμως, όταν την καλείτε με έναν generator, συμβαίνει αυτό:

```
f(n*n for n in range(4))
= []
```

Γιατί; Τι πρέπει να αλλάξετε στον ορισμό της f για να συμπεριφέρεται «σωστά» η συνάρτηση;

β) Έστω η διπλανή συνάρτηση σε ML. Ο φίλος σας που την έγραψε ισχυρίζεται ότι ελέγχει αν ένας αριθμός είναι πρώτος ή όχι, ακριβώς όπως κάναμε στο 1ο εξάμηνο.

Όμως, άγνωστο γιατί, σας προβληματίζει το ότι η συνάρτηση **`check`** ορίζεται μέσα στη συνάρτηση **`prime`**. Μπορείτε να τη βγάλετε έξω και να τη φέρετε στο ίδιο επίπεδο με την **`prime`**, έτσι ώστε να μην υπάρχει **`let`** στο πρόγραμμά σας;

Προσπαθήστε να κάνετε όσο λιγότερες αλλαγές γίνεται γιατί ο φίλος σας (και ο βαθμός σας σε αυτό το ερώτημα) θα πληγωθούν ανεπανόρθωτα...

```
fun prime 2 = true
  | prime n =
      let fun check k =
            k * k > n orelse
            n mod k <> 0 andalso
            check (k+2)
      in  n mod 2 <> 0 andalso
          check 3
      end
```

γ) Έστω το διπλανό πρόγραμμα σε μια υποθετική γλώσσα που μοιάζει με τη Java.

γ1) Τι θα εκτυπώσει το πρόγραμμα, αν η γλώσσα υλοποιεί στατική αποστολή μεθόδων; (static dispatch)

γ2) Τι θα εκτυπώσει το πρόγραμμα, αν η γλώσσα υλοποιεί δυναμική αποστολή μεθόδων; (dynamic dispatch)

```
class A {
  foo() { print(AM2); bar(); }
  bar() { print(17); }
}

class B extends A {
  foo() { print(42); bar(); }
  bar() { print(AM3); }
}

main() {
  A a = new A; a.foo();
  B b = new B; b.foo();
    a = new B; a.foo();
}
```

δ) Έστω το διπλανό πρόγραμμα σε μία γλώσσα που μοιάζει με την C++.

δ1) Τι θα εκτυπώσει το πρόγραμμα, αν η γλώσσα υλοποιεί στατικές εμβέλειες; (static/lexical scopes)

δ2) Τι θα εκτυπώσει το πρόγραμμα, αν η γλώσσα υλοποιεί δυναμικές εμβέλειες; (dynamic scopes)

```
int y = AM2;

void g(int a, int b) {
  print(y, b);
  y = a;
}

void f(int x) {
  int y = AM3;
  g(x, y);
  print(y);
}

void main() {
  f(17);
  print(ys);
}
```

3. Πέρασμα παραμέτρων (1 βαθμός)

Έστω η παρακάτω δήλωση μιας διαδικασίας σε μια υποθετική γλώσσα:

```
procedure p(value-result x, y: integer);
   <body>
```

Εάν η γλώσσα έπαυε να υποστηρίζει κλήση κατά τιμή-αποτέλεσμα και έπρεπε να χρησιμοποιήσουμε κλήση κατ' αναφορά (call by reference) στη θέση της, πώς θα έπρεπε να μετασχηματίζαμε το σώμα (body) των διαδικασιών, έτσι ώστε αυτές να εξακολουθούσαν να έχουν ακριβώς το ίδιο αποτέλεσμα; Ο μετασχηματισμός σας *θα πρέπει να δουλεύει για οποιαδήποτε* σώμα διαδικασίας. Αν το νομίζετε, η απάντηση σας μπορεί να είναι ακόμα και "Δε χρειάζεται να κάνουμε απολύτως τίποτε", αλλά η όποια απάντησή σας *θέλει επαρκή αιτιολόγηση* για την ορθότητά της.

4. Προγραμματισμός σε ML (1.5 βαθμοί)

Σε αυτό και στα επόμενα δύο προγραμματιστικά θέματα, φροντίστε ο κώδικάς σας να είναι κατανοητός και ευανάγνωστος! Προσθέστε σχόλια αν χρειάζονται!

Είχατε μία λίστα **L** αποτελούμενη από μη κενές λίστες ακεραίων αριθμών, τέτοια ώστε το πρώτο στοιχείο κάθε λίστας να ήταν ίσο με το πλήθος των επόμενων στοιχείων. Δυστυχώς δεν την έχετε πια: κάποιος την πήρε και συνένωσε όλες τις λίστες, δίνοντάς σας μία «επίπεδη» λίστα **F**.

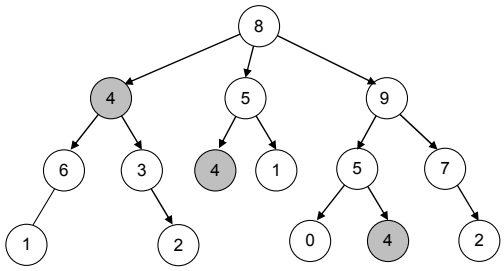
Να γραφεί σε ML μία *κομψή και αποδοτική* συνάρτηση **`reconstruct`** η οποία να δέχεται ως παράμετρο τη λίστα ακεραίων αριθμών **F** που προέκυψε με τον παραπάνω τρόπο και να ανακατασκευάζει την αρχική λίστα **L**. Παραδείγματα δίνονται παρακάτω:

```
- reconstruct [3,1,2,3,1,4,2,5,6,0,4,7,8,9,10];
val it = [[3,1,2,3],[1,4],[2,5,6],[0],[4,7,8,9,10]] : int list list

- reconstruct [];
val it = [] : int list list
```

5. Προγραμματισμός σε Prolog (0.75 + 1 = 1.75 βαθμοί)

Ένα ν-αδικό δένδρο μπορεί να αναπαρασταθεί στην Prolog από σύνθετους όρους της μορφής **`n(Data,List)`** οι οποίοι αποτελούν κόμβους που έχουν κάποια δεδομένα **`Data`**, και μια λίστα από παιδιά **`List`** από άλλους κόμβους. Οι τερματικοί κόμβοι (φύλλα) αυτού του δένδρου είναι απλά κόμβοι της μορφής **`n(Data,[])`**, για κάποια **`Data`**. Αν σας βοηθάει κάπου, μπορείτε να υποθέσετε ότι όλα τα δεδομένα είναι μη αρνητικοί ακέραιοι. Για παράδειγμα, το δένδρο της διπλανής εικόνας μπορεί να αναπαρασταθεί από τον σύνθετο όρο:




```
T = n(8, [n(4, [n(6, [n(1, [])]), n(3, [n(2, [])])]),
          n(5, [n(4, []), n(1, [])]),
          n(9, [n(5, [n(0, []), n(4, [])]), n(7, [n(2, [])])])])
```

α) Να γραφεί σε Prolog ένα *κομψό και αποδοτικό* κατηγόρημα **`max_data(Tree, Max)`**, το οποίο να ενοποιεί το **Max** με τον μέγιστο ακέραιο που υπάρχει στο δένδρο **Tree**. Για παράδειγμα, η κλήση **`max_data(n(1,[n(2,[n(17,[]),n(42,[])]),n(4,[n(5,[])])]),M)`** ενοποιεί το **`M`** με το **`42`** και η κλήση **`max_data(T,9)`** για το δένδρο του παραπάνω σχήματος πρέπει να επιτυγχάνει.

β) Να γραφεί σε Prolog ένα κατηγόρημα **`find_depth(Tree, Data, Depth)`**. Το πρώτο όρισμα είναι ένα δένδρο, το δεύτερο ένα δεδομένο προς αναζήτηση και το τρίτο ένας θετικός ακέραιος. Το κατηγόρημα πρέπει να επιτυγχάνει όταν κάποιος κόμβος του δένδρου **Tree** που βρίσκεται σε βάθος **Depth** περιέχει την πληροφορία **Data**. Θεωρούμε ότι η ρίζα βρίσκεται σε βάθος 1.

Ακολουθούν δύο παραδείγματα χρήσης, όπου θεωρούμε ότι η μεταβλητή **`T`** έχει ενοποιηθεί με τον παραπάνω όρο που αντιστοιχεί στο δένδρο του σχήματος. Στο πρώτο (αριστερά) αναζητάται η πληροφορία **`4`** στο δένδρο **`T`** (υπάρχει τρεις φορές, σε βάθη 2, 3 και 4 — βλ. γκρι κόμβους).

```
?- find_depth(T, 4, D).
D = 2 ;
D = 3 ;
D = 4 ;
false.
```

```
?- find_depth(T, X, 2).
X = 4 ;
X = 5 ;
X = 9 ;
false.
```

6. Προγραμματισμός σε Python (1 + 0.25 + 0.25 = 1.5 βαθμοί)

Δίνεται ένας κατευθυνόμενος γράφος σε αναπαράσταση με λίστα γειτνίασης **G**. Να γραφούν σε Python *κομψές και αποδοτικές* υλοποιήσεις για τις παρακάτω συναρτήσεις και να αναφερθεί η χρονική τους πολυπλοκότητα.

α) `adj_list_mat(G)`: επιστρέφει τον ίδιο γράφο σε αναπαράσταση με πίνακα γειτνίασης **M**.

β) `out_degree(M,u)`: επιστρέφει τον έξω-βαθμό (out-degree) του κόμβου **u** στο γράφο **M**.

γ) `in_degree(M,u)`: επιστρέφει τον έσω-βαθμό (in-degree) του κόμβου **u** στο γράφο **M**.

Καλή επιτυχία!