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

Επώνυμο:
Όνομα:
Αριθμός Μητρώου:

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

ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ Ι *Εαρινό 2020*

Τελικό Διαγώνισμα (Κανονική Εξέταση) #1

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 (1 βαθμός)

Παρακάτω δίνεται μια γραμματική που προσπαθεί να επιλύσει το πρόβλημα αμφισημίας του «ξεκρέμαστου else» (dangling else):

```
        <stmt> ::= if ( <expr> ) <stmt> | <matched_stmt>
<matched_stmt> ::= if ( <expr> ) <matched_stmt> else <stmt> | other
        <expr> ::= 0 | 1
```

Εξηγήστε κατά πόσο η παραπάνω προσπάθεια είναι επιτυχημένη (δηλαδή λύνει το πρόβλημα) ή όχι. Αν η απάντησή σας είναι όχι, δώστε ένα παράδειγμα πρότασης με δύο συντακτικά δένδρα.

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

Απαντήστε χωρίς αιτιολόγηση.

α) Έστω η διπλανή συνάρτηση σε ML. Ο φίλος σας που την έγραψε ισχυρίζεται ότι δέχεται μία λίστα `ls` και επιστρέφει τη λίστα όλων των μη κενών επιθεμάτων της `ls` από το μικρότερο προς το μεγαλύτερο. Δηλαδή:

```
s [1,2,3,4]
= [[4],[3,4],[2,3,4],[1,2,3,4]]
```

Όμως, κάνει λάθος γιατί όταν τη δοκιμάζετε βγάζει ένα μυστήριο type error. Διορθώστε την ώστε να δουλεύει σωστά. Όμως, μην την ξαναγράψετε από την αρχή γιατί ο φίλος σας (και ο βαθμός σας σε αυτό το ερώτημα) θα πληγωθούν ανεπανόρθωτα...

```
fun s ls =
  let fun c [] a = a
        | c (x :: r) a = c x (r :: a)
  in  c ls []
  end
```

β) Έστω το διπλανό πρόγραμμα σε Prolog. Το ερώτημα **`r(X,Y)`** δίνει 5 απαντήσεις.

Κάνοντας μία πολύ μικρή αλλαγή στον κανόνα που ορίζει το **`r/2`**, περιορίστε το έτσι ώστε να δίνει μόνο τις δύο πρώτες από αυτές τις απαντήσεις.

```
p(X) :- member(X,[2,3,5]).
q(Y) :- member(Y,[17,24,42,51,71]).

r(X,Y) :- p(X), q(Y), mod(Y,X) =:= 0.
```

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(42); bar(); }
  bar() { print(AM2); }
}

class B extends A {
  foo() { print(AM3); bar(); }
  bar() { print(17); }
}

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 x = AM3;

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

void f(int y) {
  int x = AM2;
  g(x, y);
  print(x);
}

void main() {
  f(42);
  print(x);
}
```

3. Πέρασμα παραμέτρων (4 * 0.25 = 1 βαθμός)

Έστω το παρακάτω πρόγραμμα σε κάποια υποθετική γλώσσα:

```
void p(int a, int b, int c) {
  b := b + 5;
  b := a + c + 4;
  print a, b, c;
}
```

```
void main {
  int j := AM3;
  int k := AM2;
  p(j, j, j + k);
  print j, k;
}
```

Τι θα τυπώσουν οι εκτολές **`print`** στο παραπάνω πρόγραμμα αν:

α) Όλες οι παράμετροι περνιούνται κατά τιμή (call by value)
β) Οι παράμετροι **`a`** και **`b`** περνιούνται κατ' αναφορά (call by reference) και η **`c`** κατά τιμή
γ) Οι παράμετροι **`a`** και **`b`** περνιούνται κατά τιμή αποτέλεσμα (by value-result) και η **`c`** κατά τιμή
δ) Όλες οι παράμετροι περνιούνται κατ' όνομα (call by name)

Γράψτε τις απαντήσεις σας σε αυτήν την ερώτηση σε έναν πίνακα που να μοιάζει με τον παρακάτω.

Υποερώτημα	a	b	c	j	k
α					
β					
γ					
δ					

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!

Να γραφεί σε ML μία *κομψή και αποδοτική* συνάρτηση **oddeven** η οποία να δέχεται ως παράμετρο μία λίστα ακεραίων αριθμών **L**. Η συνάρτηση αυτή πρέπει να επιστρέφει το μήκος του μακρύτερου συνεχόμενου τμήματος αυτής της λίστας, στο οποίο εναλλάσσονται περιττοί και άρτιοι αριθμοί. Παραδείγματα δίνονται παρακάτω:

```
- oddeven [1,2,3];
val it = 3 : int                    (* ολόκληρη η λίστα *)

- oddeven [6,8,17,42,37,91];
val it = 4 : int                    (* το τμήμα 8,17,42,37 *)

- oddeven [23,11,38,39,33,24,25,13,36,7,10,9,6,36];
val it = 6 : int                    (* το τμήμα 13,36,7,10,9,6 *)
```

5. Προγραμματισμός σε Prolog (1.5 βαθμοί)

Να γραφεί σε Prolog ένα *κομψό και αποδοτικό* κατηγόρημα **prime_start(L,S)**, το οποίο να αληθεύει αν και μόνο αν η λίστα **S** = **[S1,...,Sn]** είναι μία λίστα από λίστες **Si, i ≥ 0** τέτοιες ώστε κάθε μία από αυτές αρχίζει από έναν πρώτο αριθμό (2, 3, 5, 7, 11, ...), η **S1** ξεκινάει από τον πρώτο σε εμφάνιση πρώτο αριθμό της **L**, και η συνένωση όλων των **Si** λιστών μας δίνουν ένα πλήρες επίθεμα (suffix) της λίστας **L**. Στο πρόγραμμά σας, μπορείτε να υποθέσετε την ύπαρξη ενός κατηγορήματος **prime/1** το οποίο επιτυγχάνει αν το όρισμά του είναι πρώτος αριθμός. Για να πάρετε όλους τους βαθμούς, η λύση σας πρέπει να παράγει όλους τους δυνατούς τρόπους που το παραπάνω μπορεί να γίνει. Κάποια παραδείγματα ακολουθούν:

```
?- prime_start([2,1,4,5,4], S).
S = [[2,1,4], [5,4]] ;
S = [[2,1,4,5,4]] ;
false.

?- prime_start([4,6,8,42], S).
S = [].
```

```
?- prime_start([4,2,1,3,5,4], S).
S = [[2,1], [3], [5,4]];
S = [[2,1], [3,5,4]];
S = [[2,1,3], [5,4]];
S = [[2,1,3,5,4]];
false.
```

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

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

α) **has_edge_mat(M,u,v)**: ελέγχει αν υπάρχει ακμή (u, v) στο γράφο **M**.

β) **adj_mat_list(M)**: επιστρέφει τον ίδιο γράφο σε αναπαράσταση με λίστα γειτνίασης **G**. Στην επιλογή που θα κάνετε για τον ακριβή τύπο του **G**, λάβετε υπόψη ότι θα πρέπει να υλοποιήσετε αποδοτικά το ζητούμενο στο επόμενο ερώτημα.

γ) **has_edge_list(G,u,v)**: ελέγχει αν υπάρχει ακμή (u, v) στο γράφο **G** που είναι τώρα σε μορφή λίστας γειτνίασης.

Καλή επιτυχία!

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) #2

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 (0.25 + 0.5 = 0.75 βαθμοί)

Δίνεται η παρακάτω γραμματική για τις ισοσκελισμένες ακολουθίες παρενθέσεων:

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

α) Δείξτε ότι η γραμματική είναι διφορούμενη (ambiguous).
β) Κατασκευάστε μια γραμματική που να παράγει την ίδια γλώσσα με την παραπάνω αλλά χωρίς να έχει αμφισημία.

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

Απαντήστε χωρίς αιτιολόγηση.

α) Έστω η διπλανή συνάρτηση σε ML. Ο φίλος σας που την έγραψε ισχυρίζεται ότι δέχεται μία λίστα αριθμών `ls` και επιστρέφει το πλήθος των περισσότερων συνεχόμενων 42. Δηλαδή:

`c [1,2,42,42,3,42,42,42,4,42,5] = 3`

Δουλεύει, όμως δεν είναι tail recursive. Διορθώστε την ώστε να γίνει! Όμως, μην την ξαναγράψετε από την αρχή γιατί ο φίλος σας (και ο βαθμός σας σε αυτό το ερώτημα) θα πληγωθούν ανεπανόρθωτα...

```
fun c ls =
  let fun d [] n = n
        | d (42 :: t) n = d t (n + 1)
        | d (h :: t) n =
            Int.max (n, d t 0)
  in  d ls 0
  end
```

β) Έστω η διπλανή συνάρτηση σε Python.

Αν την καλέσετε με x=17 και y=8 εκτυπώνει “42”.

Μπορείτε να την κάνετε να εκτυπώσει “banana”;

Αν ναι, εξηγήστε πώς. Αν όχι, εξηγήστε γιατί.

```
def f(x,y):
  print(y+2*x)

>>> f(17,8)
42
```

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(AM3); bar(); }
  bar() { print(42); }
}

class B extends A {
  foo() { print(AM1); bar(); }
  bar() { print(AM2); }
}

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 x = AM2;

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

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

void main() {
  f(AM1s);
  print(x);
}
```

3. Πέρασμα παραμέτρων (0.3 + 0.3 + 0.4 = 1 βαθμός)

Έστω το παρακάτω πρόγραμμα σε κάποια υποθετική γλώσσα με πίνακες και τελεστές παρόμοιους με τη C/C++. (Η αρίθμηση πινάκων αρχίζει από το 0, και % είναι το υπόλοιπο της ακέραιας διαίρεσης.)

```
void foo(int x, int y, int z) {
  int t = z;
  z = y; y = x; x = t;
}
```

```
void main {
  int k = AM3 % 5, j = AM2 % 4;
  int t[5] = {1,3,2,3,1};
  foo(t[k], t[j], k);
}
```

Σε έναν πίνακα που μοιάζει με τον παρακάτω, γράψτε τις τιμές που θα έχουν οι μεταβλητές **k** και **t** μετά την κλήση της συνάρτησης **foo**, αν οι παράμετροι **x**, **y** και **z** μεταβιβάζονται:

α) κατ’ αναφορά (call by reference),

β) κατά τιμή-αποτέλεσμα (call by value-result, υποθέτοντας ότι κατά την επιστροφή η ενημέρωση των πραγματικών παραμέτρων γίνεται από τα αριστερά προς τα δεξιά), και

γ) κατ’ όνομα (call by name).

Υποερώτημα	**k**	**t**
α		
β		
γ		

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!

Να γραφεί σε ML μία *κομψή και αποδοτική* συνάρτηση **double_pairs** η οποία να δέχεται ως παράμετρο μία λίστα **L**, αποτελούμενη από θετικούς ακέραιους αριθμούς σε γνησίως αύξουσα σειρά. Η συνάρτηση αυτή πρέπει να επιστρέφει, πάλι σε αύξουσα σειρά, τα ζεύγη της μορφής (*x*, 2*x*) που απαντώνται στη λίστα, δηλαδή ζεύγη στα οποία ο δεύτερος αριθμός είναι ο διπλάσιος του πρώτου. Παράδειγμα δίνεται παρακάτω:

```
- double_pairs [1,2,3,4,5,7,8,14,21,39,42];
val it = [(1,2),(2,4),(4,8),(7,14),(21,42)] : (int * int) list
```

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

Ένα τριαδικό δένδρο κατασκευάζεται από σύνθετους όρους της μορφής **n(T1,T2,T3)** οι οποίοι αναπαριστούν κόμβους που έχουν ως **T1**, **T2** και **T3** άλλους κόμβους ή ακεραίους. Ας υποθέσουμε ότι οι μόνοι ακέραιοι που χρησιμοποιούνται είναι οι **0** και **1**. Λέμε ότι ένας τερματικός κόμβος (φύλλο) είναι κόμβος *μονής ισοτιμίας* αν περιέχει μονό αριθμό από **1**. Για παράδειγμα, ο κόμβος **n(1,1,1)** είναι μονής ισοτιμίας ενώ αντίθετα ο κόμβος **n(1,1,0)** όχι. Προσέξτε ότι οι ακέραιοι δεν είναι κόμβοι, και κατά συνέπεια δεν ορίζεται η έννοια της ισοτιμίας για αυτούς.

α) Να γραφεί σε Prolog ένα *κομψό και αποδοτικό* κατηγόρημα **triadiko_01(Tree,Zeros,Ones)**, το οποίο να επιτυγχάνει αν οι λίστες **Zeros** και **Ones** αποτελούνται από τα 0 και 1, αντίστοιχα, που υπάρχουν στο δένδρο **Tree**. Για παράδειγμα η κλήση **triadiko_01(n(n(0,1,0),1,0),Zs,Os)** πρέπει να επιστρέφει **Zs = [0,0,0]** και **Os = [1,1]**.

β) Να γραφεί σε Prolog ένα *κομψό και αποδοτικό* κατηγόρημα **count_odd_parity(Tree,Count)**, το οποίο να επιτυγχάνει αν το πλήθος των τερματικών κόμβων μονής ισοτιμίας του τριαδικού δένδρου **Tree** είναι **Count**. Δύο παραδείγματα ακολουθούν:

```
?- count_odd_parity(n(n(0,1,0),1,0), Count).
Count = 1.

?- count_odd_parity(n(n(0,1,0),n(n(0,1,1),1,0),n(1,1,1)), Count).
Count = 2.
```

6. Προγραμματισμός σε Python (1.5 βαθμοί)

Δίνεται μία συμβολοσειρά **S** αποτελούμενη από **N** μικρά γράμματα του λατινικού αλφαβήτου. Να γραφεί σε Python μία *κομψή και αποδοτική* συνάρτηση **count_substr(S, K)** που να μετράει πόσες διαφορετικές υποσυμβολοσειρές μήκους **K** γραμμάτων περιέχονται στην **S**. Τι πολυπλοκότητα έχει η συνάρτησή σας; Παραδείγματα δίνονται παρακάτω:

```
count_substr("helloworld", 3)
= 8                                  (* "hel", "ell", "llo", "low", "owo", "wor", "orl", "rld" *)

count_substr("banana", 2)
= 3                                                  (* "ba", "an", "na" *)
```


Καλή επιτυχία!

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**.

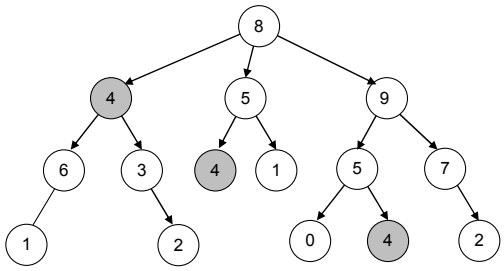
Να γραφεί σε ML μία *κομψή και αποδοτική* συνάρτηση **`reconstruct`** η οποία να δέχεται ως παράμετρο τη λίστα ακεραίων αριθμών **F** που προέκυψε με τον παραπάνω τρόπο και να ανακατασκευάζει την αρχική λίστα **L**. Παραδείγματα δίνονται παρακάτω:

```
- 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!