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

Επώνυμο:
Όνομα:
Αριθμός Μητρώου:

1	
2	
3	
4	
5	
6	
Σ	

ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ Ι ***Εαρινό 2021***

Τελικό Διαγώνισμα (Κανονική Εξέταση)

Το διαγώνισμα αυτό έχει ερωτήσεις 8 βαθμών συνολικά.

Γράψτε τις απαντήσεις σας σε ένα αρχείο κειμένου με όνομα el18042.txt ή χρησιμοποιήστε κάποιον επεξεργαστή κειμένου και μετατρέψτε το αρχείο σας σε el18042.pdf (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 ή δεν μπορείτε να το βάλετε στο pdf, 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 el18042, 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 + 0.25 = 1 βαθμός)

Δίνεται η παρακάτω γραμματική:

```
<S> ::= ( <L> )  |  a
<L> ::= <L> , <S> | <S>
```

α) Δείξτε το συντακτικό δένδρο της συμβολοσειράς **`((a,a), a, (a))`**
β) Είναι η παραπάνω γραμματική διφορούμενη (ambiguous) ή όχι; Αν ναι, δώστε ένα παράδειγμα συμβολοσειράς με δύο δένδρα. Αν όχι, δικαιολογήστε επαρκώς την απάντησή σας.
γ) Περιγράψτε με λόγια τη γλώσσα της παραπάνω γραμματικής.

2. Ερωτήσεις κατανόησης (0.25 + 0.25 + 0.5 + 0.5 = 1.5 βαθμοί)

Απαντήστε εν συντομία, χωρίς αιτιολόγηση, εκτός αν η εκφώνηση το ζητά ρητά.

α) Ένας φίλος σας θέλει να γράψει σε ML μία συνάρτηση **`common_prefix`** που να δέχεται δύο λίστες αριθμών και να βρίσκει το μακρύτερο κοινό τους πρόθεμα. Η συνάρτηση πρέπει να επιστρέφει το ζητούμενο πρόθεμα και ό,τι απομένει στις δύο λίστες μετά την αφαίρεσή του.

```
- common_prefix [1, 2, 3, 4, 5] [1, 2, 3, 5, 8];
val it = ([1,2,3],[4,5],[5,8]) : int list * int list * int list
```

Έχει ξεκινήσει να τη γράφει και μάλιστα προσπάθησε να την κάνει tail recursive. Όμως, κάπου έχει κολλήσει και ζητάει τη βοήθειά σας. Συμπληρώστε αυτά που λείπουν έτσι ώστε να λειτουργεί σωστά. Όμως, μην την ξαναγράψετε από την αρχή γιατί ο φίλος σας (και ο βαθμός σας σε αυτό το ερώτημα) θα πληγωθεί ανεπανόρθωτα...

```
fun common_prefix x y =
  let fun aux (h1 :: t1) (h2 :: t2) prefix =
            if h1 = h2 then ???
            else ???
        | aux s1 s2 prefix = ???
  in  aux x y []
  end
```

β) Ένας άλλος φίλος σας έγραψε το διπλανό κατηγόρημα σε Prolog. Δέχεται ως όρισμα μία λίστα. Ο φίλος σας ισχυρίζεται ότι το κατηγόρημα αληθεύει αν και μόνο αν η λίστα δεν έχει διπλότυπα (δηλαδή αν κάθε στοιχείο της εμφανίζεται ακριβώς μία φορά).

Διαπιστώστε αν ο ισχυρισμός του φίλου σας ισχύει. Αν ναι, εξηγήστε πώς λειτουργεί το κατηγόρημα. Αν όχι, δώστε ένα αντιπαράδειγμα και εξηγήστε ποιο είναι το πρόβλημα και ποια είναι η *ελάχιστη* αλλαγή που πρέπει να γίνει ώστε να λειτουργεί σωστά.

Προσοχή: μην ξαναγράψετε από την αρχή το κατηγόρημα γιατί και αυτός ο φίλος σας είναι πολύ ευαίσθητος.

```
unique([]).
unique([Item | Rest]) :-
  member(Item, Rest), fail.
unique([_ | Rest]) :-
  unique(Rest).
```

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(17); }
  bar() { print(AM2); foo(); }
}

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

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

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 x) {
  g(x, AM1);
  print(x);
}

void main() {
  f(AM2);
  print(x);
}
```

3. Εγγραφές Δραστηριοποίησης (1 βαθμός)

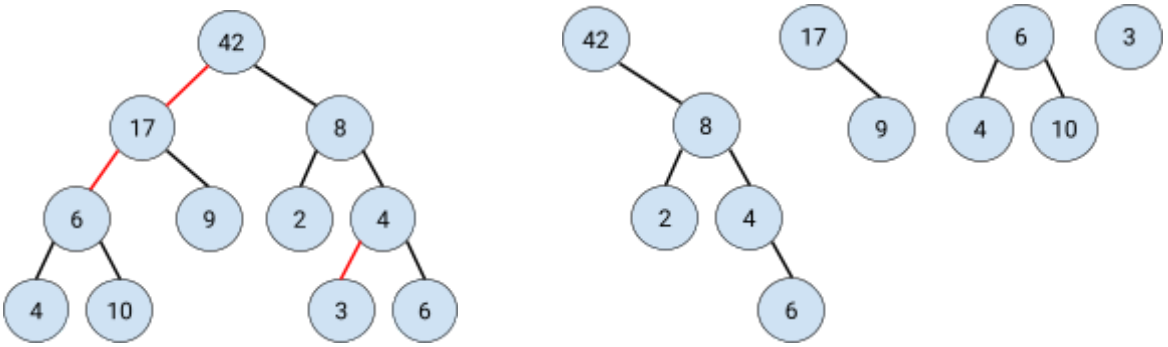
Γράψτε τη *συνοπτικότερη* συνάρτηση ML που μπορείτε να σκεφτείτε (όχι αυτή στις διαφάνειες!) η οποία δε θα λειτουργούσε σωστά αν το σύστημα υλοποίησης της γλώσσας ML χρησιμοποιούσε εγγραφές δραστηριοποίησης οργανωμένες σε στοίβα αλλά χωρίς συνδέσμους φωλιάσματος (nesting links). Εξηγήστε πώς ακριβώς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

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 ένας κατάλληλος τύπος δεδομένων **`'a tree`** για δυαδικά δέντρα, οι κόμβοι των οποίων να φέρουν ως πληροφορία στοιχεία κάποιου τύπου **`'a`**.

Έστω ένα τέτοιο δέντρο με ακέραιους αριθμούς στους κόμβους του. Λέμε ότι "κλαδεύουμε" το δέντρο αν αφαιρέσουμε όλες τις ακμές που συνδέουν κόμβους με διαφορετικό parity (υπόλοιπο ως προς δύο) και σχηματίσουμε μία λίστα με τα δέντρα που προκύπτουν. Για παράδειγμα, αν κλαδέψουμε το δέντρο του παρακάτω σχήματος (αριστερά), οι ακμές που θα αφαιρεθούν είναι σημειωμένες με κόκκινο χρώμα και το αποτέλεσμα θα είναι μία λίστα με τα δέντρα που επίσης φαίνονται (δεξιά).



Να γραφεί σε ML μία *κομψή και αποδοτική* συνάρτηση **`trim`** η οποία να δέχεται ως παράμετρο ένα τέτοιο δυαδικό δέντρο **t**. Η συνάρτηση πρέπει να επιστρέφει τη λίστα με τα δέντρα που προκύπτουν από το "κλάδεμα", όπως περιγράφηκε παραπάνω, σε οποιαδήποτε σειρά. Αν το **t** είναι το κενό δέντρο, η συνάρτησή σας θα πρέπει να επιστρέφει (προφανώς) μία λίστα με ένα μόνο κενό δέντρο.

5. Προγραμματισμός σε Prolog (0.75 + 0.75 + 0.25 = 1.75 βαθμοί)

Ένα τριαδικό δέντρο κατασκευάζεται από σύνθετους όρους της μορφής **`n(T1,T2,T3)`** οι οποίοι αναπαριστούν κόμβους που έχουν ως **`T1`**, **`T2`** και **`T3`** άλλους τέτοιους κόμβους ή μη αρνητικούς ακέραιους. Λέμε ότι ένας τερματικός κόμβος (φύλλο) της παραπάνω μορφής είναι κόμβος *περιττού αθροίσματος* αν το άθροισμα των τριών ακεραίων που περιέχει είναι περιττός αριθμός. Π.χ., ο κόμβος **`n(0,1,2)`** είναι περιττού αθροίσματος ενώ αντίθετα ο κόμβος **`n(13,17,42)`** όχι.

α) Να γραφεί σε Prolog ένα *κομψό και αποδοτικό* κατηγόρημα **`maximize(Tree,MaxTree)`**, το οποίο να επιτυγχάνει αν το δεύτερό του όρισμα είναι ένα δέντρο με δομή παρόμοια του **Tree** αλλά στο οποίο όλοι οι ακέραιοι έχουν αντικατασταθεί με τον μέγιστο ακέραιο του **Tree**. Κάποια παραδείγματα:

```
?- maximize(n(13,5,17), MaxTree).
MaxTree = n(17,17,17).

?- maximize(n(n(0,1,2),n(3,n(4,5,6),7),n(8,9,10)), MaxTree).
MaxTree = n(n(10,10,10),n(10,n(10,10,10),10),n(10,10,10)).

?- maximize(n(n(0,1,2),n(13,17,42),4), MaxTree).
MaxTree = n(n(42,42,42),n(42,42,42),42).
```

Αν θέλετε, μπορείτε να χρησιμοποιήσετε το ενσωματωμένο κατηγόρημα **`integer/1`** της Prolog το οποίο επιτυγχάνει αν το όρισμά του είναι ακέραιος, και την ενσωματωμένη αριθμητική συνάρτηση **`max/2`** η οποία αποτιμά τα δύο ορίσματά της και επιστρέφει την τιμή του αριθμητικά μεγαλύτερου από αυτά.

β) Να γραφεί σε Prolog ένα *κομψό και αποδοτικό* κατηγόρημα **`unoddsum(Tree,Term)`**, το οποίο να επιστρέφει στο δεύτερο όρισμά του τον όρο που προκύπτει αν στο δέντρο που δίνεται στο πρώτο του όρισμα αντικατασταθούν όλοι οι κόμβοι περιττού αθροίσματος με τον ακέραιο **`17`** έτσι ώστε στον τελικό όρο να μην υπάρχει κανένας κόμβος περιττού αθροίσματος. Μερικά παραδείγματα:

```
?- unoddsum(n(13,5,17), Term).
Term = n(13,5,17).

?- unoddsum(n(n(0,1,2),n(3,4,5),n(6,7,8)), Term).
Term = n(17,n(3,4,5),17).

?- unoddsum(n(n(0,1,2),n(3,n(4,5,6),7),n(8,9,10)), Term).
Term = 17.
```

Όπως φαίνεται στο τελευταίο παράδειγμα, η αντικατάσταση γίνεται αναδρομικά. Αν σας βολεύει, μπορείτε να λύσετε αυτό το υποερώτημα χωρίς αναδρομική αντικατάσταση (για 0.5 βαθμούς), αλλά θα πρέπει να αναφέρετε αν η λύση σας έχει αυτόν τον περιορισμό.

γ) Μπορούμε να γράψουμε σε ML μια συνάρτηση **`unoddsum`** που να δουλεύει σε δέντρα με την δομή των παραδειγμάτων και να επιστρέφει παρόμοια αποτελέσματα με τα παραπάνω; Δικαιολογήστε σύντομα την απάντησή σας.

6. Προγραμματισμός σε Python (1.25 βαθμοί)

Δίνεται μία λίστα **A** αποτελούμενη από **N** ακέραιους αριθμούς, και ένας ακέραιος **K**, όπου $1 \le K \le N$. Μας ενδιαφέρουν οι φέτες (slices) της λίστας που έχουν μήκος K — υπάρχουν $N-K+1$ τέτοιες φέτες. Για κάθε μία, υπολογίζουμε το άθροισμα των στοιχείων της. Θέλουμε να βρούμε το συχνότερα εμφανιζόμενο άθροισμα και πόσες φορές εμφανίζεται. Αν υπάρχουν περισσότερα συχνότερα εμφανιζόμενα αθροίσματα, επιστρέφουμε το μεγαλύτερο από αυτά.

Να γραφεί σε Python *κομψή και αποδοτική* υλοποίηση της συνάρτησης **`sliding`**(**A**, **K**) που να πραγματοποιεί το ζητούμενο και να αναφερθεί η χρονική της πολυπλοκότητα.

Παράδειγμα: αν A = [1, 4, 2, 3, 2, 1, 3, 4, 2] και K = 4, οι φέτες του πίνακα μήκους K έχουν αθροίσματα: 1+4+2+3=10, 4+2+3+2=11, 2+3+2+1=8, 3+2+1+3=9, 2+1+3+4=10, 1+3+4+2=10. Το συχνότερα εμφανιζόμενο άθροισμα είναι το 10 που εμφανίζεται τρεις φορές.

```
>>> sliding([1, 4, 2, 3, 2, 1, 3, 4, 2], 4)
(10, 3)

>>> sliding([1, 4, 2, 3, 2, 1, 3, 4, 2], 3)
(9, 2)
```

Καλή επιτυχία!