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 (Επαναληπτική 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 (0.75 βαθμοί)

Δίνεται η παρακάτω γραμματική (το αρχικό της σύμβολο είναι το <Start>):

```
<Start>  ::=  <S> a
          |   <D> b

<S>      ::=  0 <S> 1
          |   0 1

<D>      ::=  0 <D> 1 1
          |   0 1 1
```

Είναι η παραπάνω γραμματική διφορούμενη (ambiguous) ή όχι; Αν ναι, δώστε ένα παράδειγμα συμβολοσειράς με δύο δένδρα. Αν όχι, δικαιολογήστε επαρκώς την απάντησή σας.

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

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

α) Ένας φίλος σας θέλει να γράψει σε ML μία συνάρτηση **`split`** που να δέχεται ένα φυσικό αριθμό **n** και μία λίστα **L**. Η συνάρτηση πρέπει να επιστρέφει δύο λίστες **L1** και **L2** τέτοιες ώστε:

- **L1 @ L2 = L**
- length **L1** <= **n**
- length **L1** < **n** ⇒ **L2** = []

Με άλλα λόγια, η **L1** πρέπει να περιέχει τα πρώτα **n** στοιχεία της **L** και η **L2** τα υπόλοιπα, εκτός από την ειδική περίπτωση που η αρχική λίστα έχει λιγότερα από **n** στοιχεία.

```
- split 3 [1,2,3,4,5];
val it = ([1,2,3],[4,5]) : int list * int list
- split 7 [1,2,3,4,5];
val it = ([1,2,3,4,5],[]) : int list * int list
- split 0 [1,2,3,4,5];
val it = ([],[1,2,3,4,5]) : int list * int list
```

Ο φίλος σας 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 split n L =
  let fun aux 0 acc L = ???
        | aux n acc [] = ???
        | aux n acc (h :: t) = ???
  in  aux n [] L
  end
```

β) Ένας άλλος φίλος σας έγραψε το διπλανό κατηγόρημα σε Prolog. Το πρώτο του όρισμα είναι μία λίστα εισόδου. Ο φίλος σας ισχυρίζεται ότι το κατηγόρημα μετρά πόσα στοιχεία εμφανίζονται στη λίστα, αλλά μόνο μία φορά το κάθε ένα.

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.

```
count_unique([], 0).
count_unique([H|T], X) :-
  member(H, T),
  count_unique(T, X).
count_unique([H|T], Y) :-
  count_unique(T, X),
  Y is X+1.
```

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(AM1); }
  bar() { print(AM2); foo(); }
}

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

main() {
  A a = new A; a.bar();
    a = new B; a.bar();
  B b = new B; b.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 = AM1;

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

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

void main() {
  f(AM3);
  print(x);
}
```

3. Αντικειμενοστρεφής προγραμματισμός (0.3 + 0.4 + 0.3 = 1 βαθμός)

Έστω ότι πρέπει να προγραμματίσετε σε μια προστακτική αλλά μη αντικειμενοστρεφή γλώσσα προγραμματισμού όπως η C. Όμως δε θέλετε να αποχωριστείτε τα πλεονεκτήματα του αντικειμενοστρεφούς προγραμματισμού, και αποφασίζετε να πάρετε την κατάσταση στα χέρια σας.

α) Πώς θα προσομοιώσετε τις κλάσεις και τις μεθόδους;
β) Πώς θα προσομοιώσετε την κληρονομικότητα;
γ) Μπορείτε να υποστηρίξετε πολλαπλή κληρονομικότητα και πώς;

Εξηγήστε και δικαιολογήστε τις απαντήσεις σας.

4. Προγραμματισμός σε ML (1.25 βαθμοί)

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

Έστω μία λίστα αποτελούμενη από λίστες, πιθανώς διαφορετικών μηκών. Θέλουμε να κατασκευάσουμε μία λίστα που να περιέχει κατά σειρά όλα τα πρώτα στοιχεία των λιστών, στη συνέχεια όλα τα δεύτερα στοιχεία, κ.ο.κ. μέχρι να εξαντληθούν όλα τα στοιχεία όλων των λιστών.

Να γραφεί σε ML μία *κομψή και αποδοτική* συνάρτηση **`pick_in_order`** η οποία να υλοποιεί το ζητούμενο. Προσπαθήστε να την κάνετε tail recursive. Επιθυμητή πολυπλοκότητα: γραμμική ως προς το άθροισμα των μηκών των δοσμένων λιστών.

Για παράδειγμα:

```
- pick_in_order [[11,12,13,14], [21,22], [31], [], [51,52,53,54,55], [61]];
val it = [11,21,31,51,61,12,22,52,13,53,14,54,55] : int list
```

5. Προγραμματισμός σε Prolog (0.75 + 0.75 = 1.5 βαθμοί)

α) Θα εξετάσουμε την κανονικοποίηση όρων Prolog που αντιστοιχούν σε γινόμενα. Για παράδειγμα, τα γινόμενα **`(a * b) * (c * d)`** και **`(a * (b * (c * d)))`** μπορούν να κανονικοποιηθούν στην αριστερά προσεταιριστική μορφή **`a * b * c * d`** ή την ισοδύναμή της **`((a * b) * c) * d`**. Να γραφεί σε Prolog το κατηγόρημα **`norm_product(Prod,NormProd)`** το οποίο επιτυγχάνει αν το δεύτερό του όρισμα **`NormProd`** είναι ο όρος που αντιστοιχεί στην κανονική μορφή του γινομένου **`Prod`** στο πρώτο του όρισμα.

Υπόδειξη: Στην Prolog ο σύνθετος όρος **`(a * b) * c`** είναι ένας πιο φιλικός προς τον χρήστη τρόπος γραφής του όρου **`*(*(a,b),c)`**. Αν κάπου σας βοηθάει, μπορείτε να χρησιμοποιήσετε το

ενσωματωμένο κατηγόρημα **atomic(Term)** της Prolog το οποίο επιτυγχάνει αν το όρισμά του είναι ένας «απλός» (δηλ. μη σύνθετος) όρος. Δύο παραδείγματα:

```
?- atomic(a).                ?- atomic(a * b).
true                         false
```

Εναλλακτικά, μπορείτε να υποθέστε ότι υπάρχει ήδη ορισμένο ένα κατηγόρημα **is_letter/1** το οποίο επιτυγχάνει εάν το όρισμά του είναι ένα μικρό γράμμα της Αγγλικής αλφαβήτου (**a** έως **z**).

β) Σε κάποια παλιά εξέταση του μαθήματος, υπήρχε το εξής θέμα:

Στα μηνύματα SMS πολλές φορές αντικαθιστούμε ακολουθίες χαρακτήρων με άλλες μικρότερες «ισοδύναμες» ακολουθίες για συντομία. Για παράδειγμα, αν το κάνουμε για ακολουθίες τριών χαρακτήρων στα Αγγλικά, το κείμενο "`See you later, Kate`" μπορεί να γίνει "`C u l8r k8`". Σας ζητείται να γράψετε κατηγορήματα σε Prolog που να υλοποιούν αυτήν τη μετάφραση. Γράψτε το κατηγόρημα:

textify3(ListToReplace, NewCharacter, InputList, OutputList)

το οποίο αντικαθιστά όλες τις εμφανίσεις της **ListToReplace** στην **InputList** με τον χαρακτήρα **NewCharacter.** Στο υποερώτημα αυτό μπορείτε να υποθέσετε ότι η **ListToReplace** είναι πάντα μήκους τριών χαρακτήρων. Ένα παράδειγμα χρήσης είναι το:

textify3([a,t,e], '8', [s,e,e,' ',y,o,u,' ',l,a,t,e,r,' ',k,a,t,e], Answer)

που ενοποιεί τη μεταβλητή **Answer** με την λίστα **[s,e,e,' ',y,o,u,' ',l,'8',r,' ',k,'8'].**

Ένας συμφοιτητής σας είχε γράψει το παρακάτω πρόγραμμα Prolog ως απάντηση:

```
append([], B, B).
append([H|Ta], B, [H|Tc]) :- append(Ta, B, Tc).

reverse([], []).
reverse([X|Xs], Rev) :- reverse(Xs, Rev1), append(Rev1, X, Rev).

text(_, _, [], Acc, Answer) :- reverse(Acc, Answer).
text([X,Y,Z], NC, [X,Y,Z|Rest], Acc, Answer) :-
    append([NC], Acc, NAcc),
    text([X,Y,Z], NC, Rest, NAcc, Answer).
text([X,Y,Z], NC, [W|Rest], Acc, Answer) :-
    append([W], Acc, NAcc),
    text([X,Y,Z], NC, Rest, NAcc, Answer).

textify3(LtR, NC, Input, Output) :-
    text(LtR, NC, Input, [], Output).
```

Βρείτε όλα τα λάθη που πιθανόν έχει κάνει ο συμφοιτητής σας και δώστε του συμβουλές για το πώς να βελτιώσει το πρόγραμμά του (π.χ. πως να το κάνει μικρότερο ή/και πιο αποδοτικό).

6. Προγραμματισμός σε Python (1.5 βαθμοί)

Έστω μία λέξη αποτελούμενη από μικρά γράμματα του λατινικού αλφαβήτου. Ονομάζουμε **υπογραφή** της λέξης το **σύνολο** των γραμμάτων που την αποτελούν (προφανώς χωρίς να δίνουμε σημασία στα διπλότυπα γράμματα ή τη σειρά εμφάνισης των γραμμάτων). Λέξεις που έχουν την ίδια υπογραφή τις ονομάζουμε **φίλες**. Για παράδειγμα, η λέξη “hello” έχει υπογραφή {h, e, l, o}, και οι τέσσερις λέξεις “stream”, “matters”, “smarter” και “smartest” είναι φίλες.

Δεδομένου ενός κειμένου αποτελούμενου από τέτοιες λέξεις, θα ονομάζουμε **φιλικότητα** μιας λέξης το πλήθος των διαφορετικών μεταξύ τους λέξεων του κειμένου που είναι φίλες με αυτήν. Προφανώς κάθε λέξη είναι φίλη με τον εαυτό της, άρα η φιλικότητα κάθε λέξης του κειμένου είναι τουλάχιστον 1.

Να γραφεί σε Python ένα ***κομψό και αποδοτικό*** πρόγραμμα που να διαβάζει από την τυπική είσοδο (stdin) ένα κείμενο και να τυπώνει:

- πόσες διαφορετικές υπογραφές λέξεων υπάρχουν
- ποια είναι η μέγιστη φιλικότητα μεταξύ των λέξεων του κειμένου, και
- ποιες είναι οι λέξεις με τη μέγιστη φιλικότητα, σε αλφαβητική σειρά

Παράδειγμα: για το παρακάτω κείμενο

```
be smarter now and forget the rest
here is what matters
the street that crosses the stream is the longest of all streets
```

το πρόγραμμά σας πρέπει να τυπώνει:

```
15 different signature(s)
maximum friendliness is 3
friendliest words are:
matters
rest
smarter
stream
street
streets
```

Καλή επιτυχία!