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!