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!