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

Επώνυμο:
Όνομα:
Αριθμός Μητρώου:

1	1	
2	1.25	
3	1	
4	1.5	
5	1.75	
6	1.5	
Σ	8	

ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ Ι *Εαρινό 2020*

Τελικό Διαγώνισμα (Επαναληπτική Εξέταση)

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 (0.4 + 0.1 + 0.4 + 0.1 = 1 βαθμός)

Σε μία από τις εξετάσεις του Ιουνίου είχε δοθεί η παρακάτω γραμματική **G** για μια γλώσσα με εκφράσεις όπου οι αριθμητικοί τελεστές είναι σε ενθεματική (infix) μορφή:

```
<expr> ::= <expr> + <mult> | <mult>
<mult> ::= <mult> * <fact> | <fact>
<fact> ::= ( <expr> ) | a | b | c
```

Η γραμματική αυτή είναι μη διφορούμενη, π.χ. δίνει ένα μόνο δένδρο για την έκφραση `a + b * c`. Είχε ζητηθεί να προστεθούν στη γραμματική **G** δυαδικοί τελεστές **−**, **/** και **%** με τις «συνηθισμένες» προτεραιότητες και προσεταιριστικότητες ώστε η τελική γραμματική να παραμείνει μη διφορούμενη.

α) Χρησιμοποιώντας παρόμοιο συμβολισμό για τις γραμματικές με τον παραπάνω, κατασκευάστε μια γραμματική για τη γλώσσα των εκφράσεων με τους παραπάνω πέντε τελεστές (+, *, −, / και %) σε *επιθεματική (postfix)* μορφή. Για παράδειγμα, η παραπάνω έκφραση γράφεται `a b c * +` σε postfix μορφή.

β) Εξηγήστε συνοπτικά για ποιο λόγο η γραμματική της απάντησής σας είναι ή δεν είναι διφορούμενη.

γ) Επεκτείνετε την παραπάνω γραμματική **G** της εξέτασης του Ιουνίου με μοναδιαίους τελεστές (πρόσημα) + και – με τέτοιο τρόπο ώστε η γραμματική να παραμείνει μη διφορούμενη.

δ) Η γραμματική σας, επιτρέπει την έκφραση `– + – a` ή όχι;

2. Ερωτήσεις κατανόησης (5 * 0.25 = 1.25 βαθμοί)

Απαντήστε χωρίς αιτιολόγηση.

α) Έστω η διπλανή συνάρτηση σε ML. Ο φίλος σας που την έγραψε ισχυρίζεται ότι κατασκευάζει μία λίστα με τους αριθμούς από το **low** μέχρι και το **high**. Δηλαδή:

```
enum 3 10 = [3,4,5,6,7,8,9,10]
```

Δουλεύει, όμως δεν είναι tail recursive. Διορθώστε την ώστε να γίνει tail recursive και να είναι όσο το δυνατόν πιο αποδοτική σε χρόνο και μνήμη!

```
fun enum low high =
  if low <= high then
    low :: enum (low+1) high
  else
    []
```

β) Έστω το διπλανό πρόγραμμα σε μια υποθετική γλώσσα που μοιάζει με τη Java.

β1) Τι θα εκτυπώσει το πρόγραμμα, αν η γλώσσα υλοποιεί στατική αποστολή μεθόδων; (static dispatch)

β2) Τι θα εκτυπώσει το πρόγραμμα, αν η γλώσσα υλοποιεί δυναμική αποστολή μεθόδων; (dynamic dispatch)

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

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

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

γ) Έστω το διπλανό πρόγραμμα σε μία γλώσσα που μοιάζει με την C++.

γ1) Τι θα εκτυπώσει το πρόγραμμα, αν η γλώσσα υλοποιεί στατικές εμβέλειες; (static/lexical scopes)

γ2) Τι θα εκτυπώσει το πρόγραμμα, αν η γλώσσα υλοποιεί δυναμικές εμβέλειες; (dynamic scopes)

```
int z = AM3;

void g(int a, int b) {
  print(z, b);
  z = a + AM1;
}

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

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

3. Πέρασμα παραμέτρων (1 βαθμός)

Σε μία από τις εξετάσεις του Ιουνίου υπήρχε το παρακάτω θέμα για πέρασμα παραμέτρων.

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.

Ένας συμφοιτητής σας που συμμετείχε στην εξέταση ισχυρίζεται ότι δεν μπόρεσε να σκεφτεί κάποιο γενικό μετασχηματισμό που να δίνει πάντα λύση στο παραπάνω πρόβλημα, βρήκε όμως έναν μετασχηματισμό που λύνει το αντίστροφο πρόβλημα, δηλαδή με ποιο γενικό τρόπο μπορούμε να μετασχηματίσουμε τα προγράμματα μίας γλώσσας που υποστηρίζει μόνο κλήση κατ' αναφορά (call by reference) έτσι ώστε να δίνουν πάντα το ίδιο αποτέλεσμα σε μία γλώσσα που υποστηρίζει μόνο κλήση κατά τιμή-αποτέλεσμα (call by value-result).

Εσείς τι γνώμη έχετε; Μπορείτε να περιγράψετε, έστω στο περίπου, το μετασχηματισμό που σκέφτηκε ο συμφοιτητής σας (και γιατί είναι σωστός), ή να του εξηγήσετε, πιθανώς με παραδείγματα, γιατί ο μετασχηματισμός του είναι λανθασμένος;

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!

Δίνεται μία μη κενή λιστα **L** αποτελούμενη από ζεύγη ακεραίων της μορφής (key, value). Θεωρήστε ότι οι τιμές των κλειδιών είναι μη αρνητικές. Έστω ότι η μέγιστη τιμή κλειδιού που εμφανίζεται στη λίστα είναι ίση με **K**. Ζητείται να κατασκευάσετε μία λίστα αποτελούμενη από **K**+1 ακεραίους, που κάθε στοιχείο της αντιστοιχεί σε μία τιμή κλειδιού από 0 μέχρι και **K**, κατά σειρά. Το στοιχείο της ζητούμενης λίστας που αντιστοιχεί στο κλειδί key θα πρέπει να είναι ίσο με το άθροισμα των τιμών (value) που περιέχονται σε όλα τα ζεύγη της **L** που έχουν ως κλειδί το key.

Να γραφεί σε ML μία *κομψή και αποδοτική* συνάρτηση **`bidlist`** η οποία να δέχεται ως παράμετρο τη λίστα **L** και να επιστρέφει το ζητούμενο. Για παράδειγμα:

```
- bidlist [(3,17),(0,42),(2,7),(3,25),(2,10),(7,3)];
val it = [42,0,17,42,0,0,0,3] : int list
```

Το πρώτο στοιχείο του αποτελέσματος είναι 42 γιατί υπάρχει μόνο ένα ζεύγος (0, 42) με κλειδί 0. Το δεύτερο είναι 0 γιατί δεν υπάρχει ζεύγος με κλειδί 1. Το τέταρτο στοιχείο είναι 42 γιατί υπάρχουν δύο ζεύγη (3, 17) και (3, 25) με κλειδί 3 και 17+25=42. Η μέγιστη τιμή κλειδιού εδώ είναι **K**=7.

Προσοχή: Σε αυτό το θέμα δεν πρέπει να χρησιμοποιήσετε άλλη δομή δεδομένων εκτός των λιστών.

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

α) Οι φυσικοί αριθμοί μπορούν να αναπαρασταθούν στην Prolog με τους όρους `0`, `s(0)`, `s(s(0))`,... — όπου το `s` σημαίνει επόμενος (successor) — και να κατασκευαστούν από το κατηγόρημα:

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).
```

Χρησιμοποιώντας την παραπάνω αναπαράσταση (δηλαδή χωρίς να μετατρέψετε τους αριθμούς σε ακέραιους, στην Prolog), γράψτε το κατηγόρημα **`gcd(X, Y, GCD)`** το οποίο να ενοποιεί το τρίτο του όρισμα **`GCD`** με τον μέγιστο κοινό διαιρέτη των δύο πρώτων, **`X`** και **`Y`**. Κάποια παραδείγματα:

```
?- gcd(s(s(s(s(0)))), s(s(s(s(s(s(0)))))), GCD).
GCD = s(s(0)) ;
false.

?- gcd(s(s(s(s(0)))), s(s(s(s(s(s(s(s(0))))))))), GCD).
GCD = s(s(s(s(0)))) ;
false.

?- gcd(s(s(s(0))), s(s(s(s(s(s(s(s(s(0))))))))), GCD).
GCD = s(0) ;
false.
```

β) Το παρακάτω κατηγόρημα σε Prolog επιτυγχάνει αν το δεύτερό του όρισμα είναι μια λίστα που προκύπτει από την αναδιάταξη (permutation) των στοιχείων της λίστας στο πρώτο του όρισμα.

```
permutation([], []).
permutation([X|Xs], Ys1) :- permutation(Xs, Ys), select(X, Ys1, Ys).
```

Λέμε ότι μια αναδιάταξη **Y** μιας λίστας **X** είναι *αναδιάταξη περιττής ισοτιμίας* (*odd parity*) αν το πλήθος των ζευγών στοιχείων (a, b) που εμφανίζονται με αυτή τη σειρά στη λίστα **X** αλλά με αντίστροφη σειρά στη λίστα **Y** — δηλαδή ως (b, a) — είναι περιττό. Θεωρήστε ότι οι λίστες έχουν διακριτά στοιχεία.

Να γραφεί σε Prolog το κατηγόρημα **`odd_permutation/2`** που επιτυγχάνει αν το δεύτερό του όρισμα είναι μια αναδιάταξη περιττής ισοτιμίας του πρώτου του ορίσματος. Κάποια παραδείγματα:

```
?- odd_permutation([1,2], Y).
Y = [2,1] ;
false.

?- odd_permutation([1,2,3], Y).
Y = [2,1,3] ;
Y = [1,3,2] ;
Y = [3,2,1] ;
false.

?- Y = [1|_], odd_permutation([1,2,3,4], Y).
Y = [1,3,2,4] ;
Y = [1,2,4,3] ;
Y = [1,4,3,2] ;
false.
```

Το πρώτο παράδειγμα είναι προφανές. Στο δεύτερο παράδειγμα, η αναδιάταξη **`[2,1,3]`** είναι περιττής ισοδυναμίας γιατί το μόνο ζεύγος στοιχείων που αλλάζουν διάταξη είναι τα 1 και 2, ενώ η **`[3,2,1]`** είναι περιττής ισοδυναμίας διότι υπάρχουν συνολικά τρία ζεύγη στοιχείων που αλλάζουν διάταξη. Στο τελευταίο παράδειγμα, με την ενοποίηση **`Y = [1|_]`** έχουμε κρατήσει μόνο τις αναδιατάξεις περιττής ισοτιμίας της λίστας **`[1,2,3,4]`** που αρχίζουν από **`1`**. Αλλιώς, υπάρχουν συνολικά 12 τέτοιες αναδιατάξεις και θα έπαιρνε πολύ χώρο να τις δείχναμε όλες.

Υπόδειξη: Φυσικά, μπορείτε να χρησιμοποιήσετε το κατηγόρημα **`permutation/2`** στη λύση σας.

6. Προγραμματισμός σε Python (1.25 + 0.25 = 1.5 βαθμοί)

Δίνεται μία λίστα **A** αποτελούμενη από ακέραιους αριθμούς και ένας ακέραιος αριθμός **K**. Ζητείται να υπολογιστεί πόσες μη κενές φέτες (slices) της λίστας A υπάρχουν, τέτοιες ώστε το άθροισμα των στοιχείων τους να είναι ίσο με **K** — δηλαδή **`sum(A[i:j]) = K`**.

α) Να γραφεί σε Python μία *κομψή και αποδοτική* συνάρτηση **`countsumk(A,K)`** που να επιστρέφει το ζητούμενο.

β) Να αναφερθεί η *χρονική* και η *χωρική* πολυπλοκότητα της υλοποίησης που δώσατε ως απάντηση στο υποερώτημα (α).