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

Επώνυμο:	good
Όνομα:	enough
Αριθμός Μητρώου:	

1	0.8	
2	1.2	
3	1	
4	2	
5	1	
6	1	
7	1	
Σ	8	

ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ Ι *Εαρινό 2019*

Τελικό Διαγώνισμα (Κανονική Εξέταση)

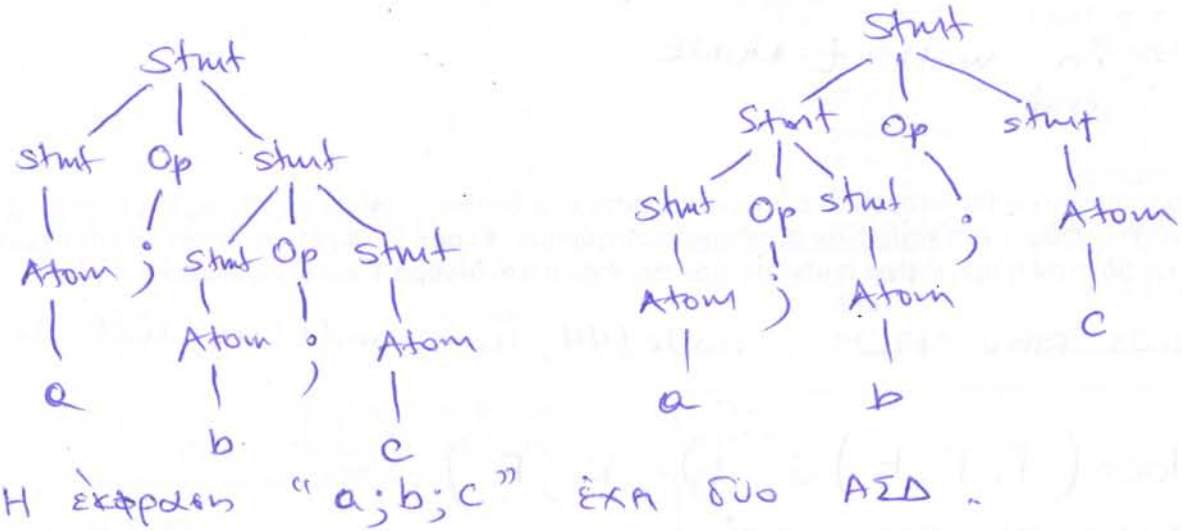
Το διαγώνισμα αυτό έχει ερωτήσεις 8 βαθμών συνολικά.

1. Γραμματικές (0.2 + 0.6 = 0.8 βαθμοί)

Έστω η παρακάτω γραμματική χωρίς συμφραζόμενα για μία γλώσσα ταυτόχρονου προγραμματισμού. Ο τελεστής || (παράλληλη εκτέλεση) έχει τη μικρότερη προτεραιότητα, έπεται ο τελεστής ; (ακολουθιακή εκτέλεση) και στη συνέχεια ο τελεστής **unless** που έχει τη μεγαλύτερη προτεραιότητα. Όλοι οι τελεστές είναι αριστερά προσεταιριστικοί. Τα **a**, **b**, **c** παριστάνουν ατομικές ενέργειες. Τα άγκιστρα χρησιμεύουν για την ομαδοποίηση των εντολών.

```
<Stmt>  ::=  <Atom>  |  { <Stmt> }  |  <Stmt> <Op> <Stmt>
<Atom>  ::=  a | b | c
<Op>    ::=  || | ; | unless
```

α) Δείξτε ότι η παραπάνω γραμματική είναι διφορούμενη.



Η έκφραση "a;b;c" έχει δύο ΑΣΔ.

β) Τροποποιήστε τη γραμματική ώστε να μην είναι διφορούμενη και οι τελεστές να έχουν τη σωστή προτεραιότητα και προσεταιριστικότητα.

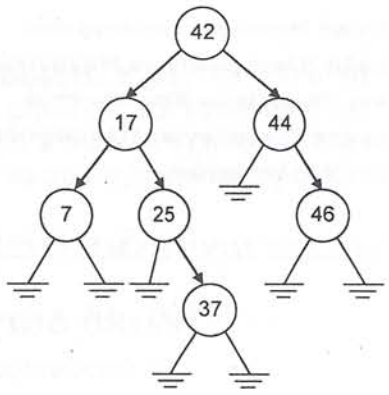
$\langle Stmt \rangle ::= \langle Stmt \rangle \,||\, \langle Stmt_2 \rangle \mid \langle Stmt_2 \rangle$

$\langle Stmt_2 \rangle ::= \langle Stmt_2 \rangle \,;\, \langle Stmt_3 \rangle \mid \langle Stmt_3 \rangle$

$\langle Stmt_3 \rangle ::= \langle Stmt_3 \rangle \text{ unless } \langle Stmt_4 \rangle \mid \langle Stmt_4 \rangle$

$\langle Stmt_4 \rangle ::= a \mid b \mid c \mid \{ \langle Stmt \rangle \}$

2. Δένδρα σε ML και Prolog (0.6 + 0.6 = 1.2 βαθμοί)



Δίνεται ένα δένδρο *δυαδικής αναζήτησης* με πληροφορία ακέραιους αριθμούς. Δίνεται επίσης ένας ακέραιος **K**. Ζητείται η μέγιστη τιμή που περιέχεται στο δένδρο η οποία δεν υπερβαίνει το **K**. Προσέξτε ότι μπορεί να μην υπάρχει τέτοια τιμή. Για παράδειγμα, για το δένδρο του σχήματος και για K=30, η ζητούμενη τιμή είναι 25.

α) Ορίστε σε ML έναν κατάλληλο τύπο δεδομένων `'a tree` για τα δυαδικά δένδρα. Στη συνέχεια, γράψτε μία *κομψή και αποδοτική* συνάρτηση `floor` που να κάνει τα παραπάνω. Ο τύπος της πρέπει να είναι: `int tree -> int -> int option`. (Θυμηθείτε: `datatype 'a option = NONE | SOME of 'a`)

```
datatype 'a tree = Leaf | Node of 'a * 'a tree * 'a tree

fun floor t k =
    let fun walk Leaf sofar = sofar
          | walk (Node(x,l,r)) sofar =
              if k=x then SOME x
              else if k<x then walk l sofar
              else walk r (SOME x)
    in walk t NONE
    end
```

b) Περιγράψτε πώς θα μπορούσατε να αναπαραστήσετε δυαδικά δένδρα ως όρους της Prolog. Στη συνέχεια, γράψτε ένα *κομψό και αποδοτικό* κατηγόρημα `floor(T,K,F)` το οποίο να επιτυγχάνει αν και μόνο αν η τιμή **F** είναι η μέγιστη που περιέχεται στο δένδρο **T** και δεν υπερβαίνει το **K**.

όπως στην ML : `node(44, leaf, node(46, leaf, leaf))`

```
floor(node(X,L,R),K,F) :-
    ( K =:= X -> F = X
    ; K < X -> floor(L,K,F)
    ; floor(R,K,F)
    ; F = X
    ).
```

3. Συμπερασμός τύπων στην ML (4 * 0.25 = 1 βαθμός)

Συμπληρώστε τους τύπους των παρακάτω συναρτήσεων:

```
fun foo x y     = x * y x + 1
fun bar x y     = x y + y 42
fun doh (x, y) = y (hd x) x
fun ugh x y z   = x y (z y)
```

	Τύπος
foo	int→(int→int)→int
bar	((int→int)→int)→(int→int)→int
doh	'a list * ('a→'a list→'b)→'b
ugh	('a→'b→'c)→'a→('a→'b)→'c

4. Ερωτήσεις πολλαπλής επιλογής (8 * 0.25 = 2 βαθμοί)

Κάθε λάθος απάντηση αφαιρεί 0.1 βαθμό από αυτό το θέμα.

Έστω το διπλανό πρόγραμμα σε Python.

```
a = b = 1
c = [b]

def f(x):
  print(x)
  a = 3
  x = 2
  print(x)

def q1():
  f(a)
  print(a)

def g(x):
  global b
  b = 3
  print(x[0])
  x[0] = 2
  print(c[0])

def q2():
  g(c)
  print(c[0)
```

α1) Τι θα εκτυπώσει η κλήση `q1()`;

(A) 1 2 1 B. 1 2 2

Γ. 1 2 3 Δ. άλλο

α2) Τι θα εκτυπώσει η κλήση `q2()`;

A. 1 1 1 (B) 1 2 2

Γ. 3 2 2 Δ. άλλο

β) Έστω το διπλανό πρόγραμμα σε Prolog. Πόσα ζεύγη απαντήσεων (X, Y) θα δώσουν οι παρακάτω ερωτήσεις-στόχοι;

```
p(17).
p(42).
p(7).

qa(X,Y) :- p(X), p(Y), X >= Y.

qb(X,Y) :- p(X), !, p(Y), X >= Y.

pc(X)   :- p(X), !.

qc(X,Y) :- pc(X), pc(Y), X >= Y.
```

β1) `qa(X,Y).`

A 3 (B) 6 Γ. 9 Δ. άλλο

β2) `qb(X,Y).`

A. 1 (B.) 2 Γ. 3 Δ. άλλο

β3) `qc(X,Y).`

(A.) 1 B. 2 Γ. 3 Δ. άλλο

γ) Η υλοποίηση μίας γλώσσας προγραμματισμού αποθηκεύει στο σωρό κατά την εκτέλεση των προγραμμάτων πολλά μικρά αντικείμενα (2 λέξεων), τα οποία ζουν πολύ λίγο. Ποιο μηχανισμό συλλογής σκουπιδιών θα προτείνατε, αν το κύριο μέλημα ήταν η ταχύτητα συλλογής;

A. Μαρκάρισμα και σκούπισμα (mark and sweep) (B.) Αντιγραφής (copying)

Γ. Μέτρημα αναφορών (reference counting) Δ. Οποιονδήποτε από τους A και B

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

```
class A {
  foo() { print(1); bar(); }
  bar() { print(2); }
}
class B extends A {
  foo() { print(3); bar(); }
  bar() { print(4); }
}
main() {
  A a = new A; a.foo();
    a = new B; a.foo();
  B b = new B; b.foo();
}
```

δ1) Αν η γλώσσα υλοποιεί στατική αποστολή μεθόδων (static dispatch), το πρόγραμμα θα εκτυπώσει:

(A.) 1 2 1 2 3 4 Β. 1 2 1 4 3 4

Γ. 1 2 3 2 3 4 Δ. 1 2 3 4 3 4

δ2) Αν η γλώσσα υλοποιεί δυναμική αποστολή μεθόδων (dynamic dispatch), το πρόγραμμα θα εκτυπώσει:

Α. 1 2 1 2 3 4 Β. 1 2 1 4 3 4

Γ. 1 2 3 2 3 4 (Δ.) 1 2 3 4 3 4

5. Πέρασμα παραμέτρων (0.5 + 0.5 = 1 βαθμός)

Έστω μία γλώσσα συναρτησιακού προγραμματισμού, για την οποία δεν είμαστε σίγουροι με ποιο τρόπο περνούν οι παράμετροι των συναρτήσεων. Μπορεί αυτό να γίνεται είτε **κατ' όνομα** (call by name), είτε **κατ' ανάγκη** (call by need), αλλά δε γνωρίζουμε ποιο από τα δύο.

α) Γράψτε ένα πρόγραμμα που θα σας επιτρέψει να διαπιστώσετε με ποιο τρόπο περνούν οι παράμετροι. Εξηγήστε σύντομα τι περιμένετε σε κάθε περίπτωση. Μπορείτε να υποθέσετε ότι η γλώσσα έχει παρενέργειες (π.χ. ανάθεση σε μεταβλητές, εκτύπωση, κ.λπ.)

Έστω σύνταξη σαν της ML.

```
fun f x = x + x
fun test () = f (print "BOO!" ; 1)
```

Αν το μήνυμα τυπωθεί δύο φορές ⇒ CBName

Αν τυπωθεί μόνο μία ⇒ CBNeed

β) Όπως το (α) αλλά η γλώσσα είναι αγνή συναρτησιακή, επομένως δεν υπάρχουν παρενέργειες.

Ομοίως, σύνταξη σαν της ML.

```
fun f 0 x = x
  | f n x = f (n-1) (x+x)

fun test () = f 42 1
```

Η κλήση της f n x με CBName

κάνει συνολικά $O(2^n)$ προσθέσεις.

Αντίθετα με CBNeed κάνει $O(n)$ ».

Τρέχουμε το test() και βλέπουμε πότε τελειώνει.

(Αυτό δεν είναι μόνο "good enough")

6. Προγραμματισμός σε ML (1 βαθμός)

Να γραφεί σε ML μία *κομψή και αποδοτική* συνάρτηση **allsubseq** η οποία να δέχεται ως παράμετρο μία λίστα **L** και να επιστρέφει μία λίστα αποτελούμενη από όλες τις υποακολουθίες της **L**. (Η λίστα **S** είναι υπακολουθία της λίστας **L** αν η **S** προκύπτει από την **L** με την αφαίρεση μηδενός ή περισσότερων στοιχείων.) Η σειρά με την οποία θα εμφανίζονται οι υποακολουθίες στο αποτέλεσμα δεν είναι σημαντική. Παραδείγματα δίνονται παρακάτω:

```
- allsubseq [1,2,3];
val it = [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]] : int list list

- allsubseq ["one", "two"];
val it = [[],["one"],["two"],["one","two"]] : string list list
```

Φροντίστε ο κώδικάς σας να είναι κατανοητός και ευανάγνωστος! Προσθέστε σχόλια αν χρειάζονται!

```
fun allsubseq xs =
    let fun nonempty [] = []
          | nonempty (x :: xs) =
              let fun walk [] acc = rev acc
                    | walk (ys :: yss) acc =
                        walk yss ((x::ys) ::
                                  ys :: acc)
              in walk (nonempty xs) [[x]]
              end
    in [] :: nonempty xs
    end
```

Η nonempty επιστρέφει όλες τις μη κενές υποακολουθίες. Χρησιμοποιεί τη walk που μαζεύει στον accumulator όλες τις μη κενές υπακολουθίες του xs (ys), και άλλη μια φορά με το x μπροστά (x::ys). Ο accumulator ξεκινά από το [x] μόνο του. Το rev είναι περιττό, αν δε μας ενδιαφέρει η σειρά εμφάνισης να είναι όπως στην εκφώνηση.

7. Προγραμματισμός σε Prolog (1 βαθμός)

Να γραφεί σε Prolog ένα <u>*κομψό και αποδοτικό*</u> κατηγόρημα **incsubseq(L,K,S)**, το οποίο να αληθεύει αν και μόνο αν η λίστα **S** είναι μία υποακολουθία της λίστας **L** μήκους **K** και τα στοιχεία της είναι σε αύξουσα σειρά. (Η λίστα **S** είναι υπακολουθία της λίστας **L** αν η **S** προκύπτει από την **L** με την αφαίρεση μηδενός ή περισσότερων στοιχείων.) Η λύση σας πρέπει να συμπεριφέρεται ως εξής:

```
?- incsubseq([2,1,4,5,4], 3, S).
S = [2,4,5] ;
S = [2,4,4] ;
S = [1,4,5] ;
S = [1,4,4] ;
false.
```

```
?- incsubseq([3,2,1], 2, S).
false.

?- incsubseq([3,4,1,2], 3, S).
false.
```

Φροντίστε ο κώδικάς σας να είναι κατανοητός και ευανάγνωστος! Προσθέστε σχόλια αν χρειάζονται!

```
incsubseq(L,K,S) :-
    length(S,K),
    subseq(S,L),
    increasing(S).

subseq([],_).
subseq([X|T1],[X|T2]) :- subseq(T1,T2).
subseq([X|T1],[_|T2]) :- subseq([X|T1],T2).

increasing([]).
increasing([_]).
increasing([H1,H2|T]) :- H1 =< H2,
                         increasing([H2|T]).
```

Αυτή είναι η "good enough".

```
incsubseq(L,K,S) :- incsubseq(L,K,S,0).
incsubseq(L,K,S,Min) :-
  ( K =:= 0 -> S = []
  ; K > 0, L = [H|T] ->
    ( H >= Min, Km is K-1, S = [H|Sm1],
                  incsubseq(T,Km1,Sm1,H)
    ; incsubseq(T,K,S,Min)
    )
  ).
```

Αυτή είναι για το 100%

/* Υπόθεση: μη αρνητικοί ακέραιοι */