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

Επώνυμο:	(more than) good enough
Όνομα:	
Αριθμός Μητρώου:	☺

1	1	1
2	1	1
3	1	1
4	1	1
5	2	2
6	1	1
7	1	0,9
Σ	8	7,9

ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ Ι *Εαρινό 2018*

Τελικό Διαγώνισμα (Κανονική Εξέταση)

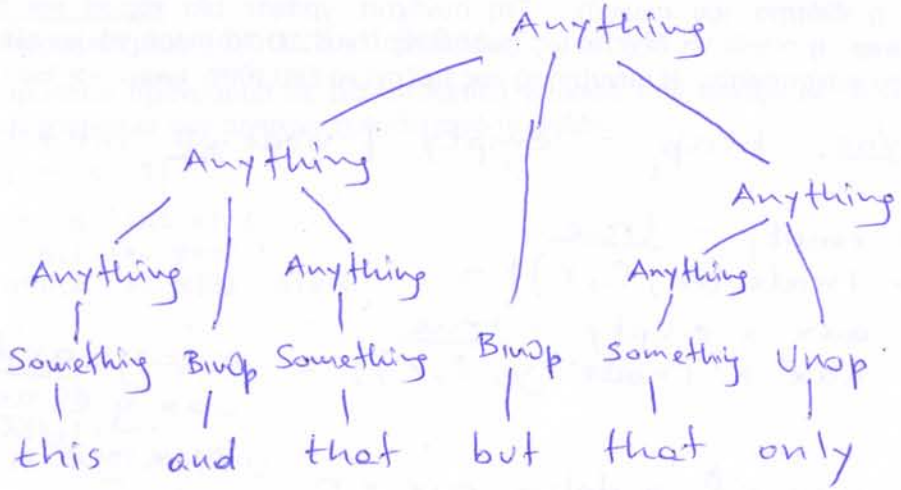
Το διαγώνισμα αυτό έχει ερωτήσεις 8 βαθμών συνολικά.

1. Γραμματικές (0.2 + 0.2 + 0.6 = 1 βαθμός)

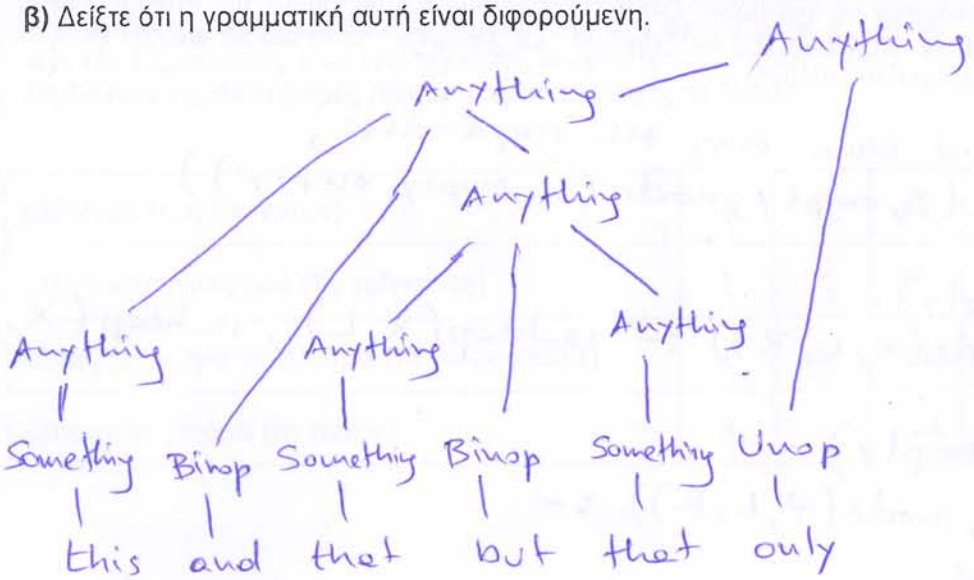
Έστω η παρακάτω γραμματική χωρίς συμφραζόμενα η οποία γεννά ένα υποσύνολο των εκφράσεων μίας συναρτησιακής γλώσσας προγραμματισμού σαν την ML.

```
<Anything>   ::= <Something>  |  ( <Anything> )
               |  <Anything> <BinOp> <Anything> | <Anything> <UnOp>
<Something>  ::= this | that
<BinOp>      ::= and | but
<UnOp>       ::= only
```

α) Δώστε ένα συντακτικό δένδρο για τη συμβολοσειρά: **`this and that but that only`**



β) Δείξτε ότι η γραμματική αυτή είναι διφορούμενη.



Η ίδια συμβολοσειρά εξόδου έχει κι άλλο συντακτικό δένδρο.

γ) Τροποποιήστε τη γραμματική ώστε να μην είναι διφορούμενη και οι τελεστές να έχουν τη σωστή προτεραιότητα και προσεταιριστικότητα. Οι τελεστές με δύο τελούμενα πρέπει να είναι δεξιά προσεταιριστικοί. Ο πίνακας προτεραιότητας των τελεστών είναι: **only** > **but** > **and**.

<Anything> ::= <A> | <A> and <Anything>
<A> ::= <B> | <B> but <A>
<B> ::= <C> | <B> only
<C> ::= this | that | (<Anything>)

2. Σωροί σε ML και Prolog (0.5 + 0.5 = 1 βαθμός)

Ένας δυαδικός σωρός ελαχίστου ορίζεται (για αυτή την άσκηση) ως ένα δυαδικό δένδρο που περιέχει ως πληροφορία ακέραιους αριθμούς και έχει την εξής ιδιότητα: δεν υπάρχει κόμβος που η τιμή του να είναι μικρότερη από την τιμή του πατέρα του.

α) Ορίστε σε ML έναν κατάλληλο τύπο δεδομένων **heap** για τον σωρό (που φυσικά δεν θα εγγυάται ότι ικανοποιείται η ιδιότητα του σωρού). Στη συνέχεια, γράψτε μία *κομψή και αποδοτική* συνάρτηση **isHeap** η οποία να δέχεται ως παράμετρο έναν τέτοιο σωρό και να ελέγχει αν η ιδιότητα του σωρού ικανοποιείται. Η συνάρτησή σας πρέπει να έχει τύπο **heap -> bool**.

```
datatype heap = empty | node of int * heap * heap

isHeap empty = true
isHeap (node (x, l, r)) =
  let aux x empty = true
      aux x (node (y, l, r)) = x <= y andalso
                               aux y l andalso
                               aux y r
  in aux x l andalso aux x r
  end
```

b) Περιγράψτε πώς θα μπορούσατε να αναπαραστήσετε τέτοιους σωρούς ως όρους της Prolog. Στη συνέχεια, γράψτε ένα *κομψό και αποδοτικό* κατηγόρημα **is_heap/1** το οποίο να ελέγχει αν η παράμετρός του είναι ένας έγκυρος σωρός.

Το αναπαριστώ όπως στην ML παραπάνω,
πχ. node(3, empty, node(4, empty, empty))

```
is_heap(empty).
is_heap(node(X, L, R)) :- is_heap(X, L), is_heap(X, R).

is_heap(X, empty).
is_heap(X, node(Y, L, R)) :-
    X =< Y,
    is_heap(Y, L),
    is_heap(Y, R).
```

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

Συμπληρώστε τους τύπους των παρακάτω συναρτήσεων:

```
fun foo x y     = y (x + 42) * 17
fun bar x y z   = x z (y z z)
fun doh (x, y) = x y + y
fun ugh x y z   = x (z :: y)
```

	Τύπος
foo	$int \rightarrow (int \rightarrow int) \rightarrow int$
bar	$(\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow \alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \gamma$
doh	$(int \rightarrow int) * int \rightarrow int$
ugh	$(\alpha\ list \rightarrow \beta) \rightarrow \alpha\ list \rightarrow \alpha \rightarrow \beta$

4. Πέρασμα παραμέτρων (4 * 0.25 = 1 βαθμός)

Έστω το παρακάτω πρόγραμμα σε μια υποθετική γλώσσα που μοιάζει με τη C. Προσέξτε ότι η αρίθμηση των στοιχείων των πινάκων ξεκινάει από το μηδέν.

```
int A[2] = {4, 7};

void f(int x, int y) {
    x++; A[1]++; y++;
    printf(x, y, A[0], A[1]);
}

void main() {
  int k = 0;
  f(k, A[k]);
  print(k, A[0], A[1]);
}
```

Συμπληρώστε τον πίνακα με τις επτά τιμές που θα εκτυπώσει το πρόγραμμα για καθεμία από τις παρακάτω τεχνικές περάσματος παραμέτρων της συνάρτησης `f`. Αν σας κάνει διαφορά και για τις τεχνικές περάσματος που έχει σημασία, θεωρήστε ότι οι πραγματικές παράμετροι υπολογίζονται και λαμβάνουν τις τελικές τιμές τους από αριστερά προς τα δεξιά.

κλήση με τιμή (by value)	1	5	4	8	0	4	8
κλήση κατ' αναφορά (by reference)	1	5	5	8	1	5	8
κλήση με τιμή-αποτέλεσμα (by value-result)	1	5	4	8	1	5	8
κλήση κατ' όνομα (by name)	1	9	4	9	1	4	9

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

Κάθε λάθος απάντηση αφαιρεί 0.1 βαθμό από αυτό το θέμα.

α) Έστω το διπλανό πρόγραμμα σε μια υποθετική γλώσσα που μοιάζει με τη C.

```
int a = 17;
int f() { print(a); }
int g() {
  int a = 42;
  f();
}
void main() {
  g();
  f();
}
```

α1) Αν η γλώσσα υλοποιεί στατικές εμβέλειες, το πρόγραμμα θα εκτυπώσει:

A. 17, 17 B. 17, 42 Γ. 42, 17 Δ. 42, 42

α2) Αν η γλώσσα υλοποιεί δυναμικές εμβέλειες, το πρόγραμμα θα εκτυπώσει:

A. 17, 17 B. 17, 42 Γ. 42, 17 Δ. 42, 42

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

```
class B {
  int x;
  B() { x = 42; }
  void f() { print(x); }
};
class D extends B {
  int x;
  D() { x = 17; }
  void f() { print(x); }
}
void main() {
  B b = new B(); b.f();
  B c = new D(); c.f();
  D d = new D(); d.f();
}
```

β1) Αν η γλώσσα υλοποιεί στατική αποστολή μεθόδων (static dispatch), το πρόγραμμα θα εκτυπώσει:

A. 42, 17, 17 B. 42, 17, 42

Γ. 42, 42, 17 Δ. 42, 42, 42

β2) Αν η γλώσσα υλοποιεί δυναμική αποστολή μεθόδων (dynamic dispatch), το πρόγραμμα θα εκτυπώσει:

A. 42, 17, 17 B. 42, 17, 42

Γ. 42, 42, 17 Δ. 42, 42, 42

γ) Έστω το διπλανό πρόγραμμα σε Prolog. Πόσες απαντήσεις θα δώσουν οι παρακάτω ερωτήσεις-στόχοι;

```
p(X)  :- member(X, [1, 2, 3]).
qa(Z) :- p(X), p(Y), Z is X+Y.
qb(Z) :- p(X), !, p(Y), Z is X+Y.
pc(X) :- p(X), !.
qc(Z) :- pc(X), pc(Y), Z is X+Y.
```

γ1) `qa(Answer).`

A. 1 B. 3 Γ. 5 Δ. άλλο

γ2) `qb(Answer).`

A. 1 B. 3 Γ. 5 Δ. άλλο

γ3) `qc(Answer).`

A. 1 B. 3 Γ. 5 Δ. άλλο

δ) Τι θα εκτυπώσει το διπλανό πρόγραμμα Python;

```
A = [x+1 for x in range(10)]
B = [10-x for x in range(10)]
L = [A[i] + B[i] for i in range(10)]
print(sum(L[3:5]))
```

A. 22 B. 33 Γ. 110 Δ. άλλο

6. Προγραμματισμός σε ML (1 βαθμός)

Να γραφεί σε ML μία *κομψή και αποδοτική* συνάρτηση **itermap** η οποία, όπως η **map**, να δέχεται ως παραμέτρους μία συνάρτηση **f** και μία λίστα **lst** από n ≥ 0 τιμές: $[v_0, v_1, \ldots, v_{n-1}]$. Εν αντιθέσει όμως με τη **map**, η **itermap** θα πρέπει να εφαρμόζει την **f** τόσες φορές πάνω σε κάθε στοιχείο της λίστας όσες η θέση του στοιχείου. Δηλαδή, θα πρέπει:

itermap f $[v_0, v_1, \ldots, v_{n-1}] = [v_0, f(v_1), f(f(v_2)), f(f(f(v_3)), \ldots]$

Το τελευταίο στοιχείο της λίστας που επιστρέφεται θα πρέπει να είναι το $f(f(\ \ldots\ (f(v_{n-1}))\ \ldots))$ όπου η f θα έχει εφαρμοστεί n-1 φορές. Παραδείγματα δίνονται παρακάτω:

```
- itermap (fn x => x+1) [1, 2, 3, 4, 5, 6, 7];
val it = [1,3,5,7,9,11,13] : int list

- itermap (fn x => 2*x) [1, 1, 1, 1, 1, 1, 1];
val it = [1,2,4,8,16,32,64] : int list

- itermap (fn s => s^s) ["a", "b", "c"];
val it = ["a","bb","cccc"] : string list
```

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

```
fun itermap f l =
    let fun aux g [] = []
          | aux g (h::t) =
                gh :: aux (f o g) t
    in  aux (fn x => x) l
    end

(* Στην g, κρατάμε τη συνάρτηση
   που θα εφαρμόσουμε στο 1ο στοιχείο
   της λίστας που δίνεται ως 2η παράμετρο.
   Το "o" στο "f o g" είναι σύνθεση
   συναρτήσεων. Κατά τα άλλα
   ευανάγνωστο, I'm taking my chances. *)
```

Εναλλακτικά:

```
fun itermap f [] = []
  | itermap f (h::t) = h :: itermap f (map f t)
(* αλλα παίρνει -0.1 γιατί φτιάχνει η λίστα *)
```

7. Προγραμματισμός σε Prolog (1 βαθμός)

Να γραφεί σε Prolog ένα *κομψό και αποδοτικό* ντετερμινιστικό κατηγόρημα (δηλαδή μια συνάρτηση) **middle** το οποίο ομαδοποιεί τα στοιχεία μιας λίστας από ακεραίους σε υπο-λίστες, οι οποίες αρχίζουν από τη μέση της. Με άλλα λόγια, ο ένας ή οι δύο ακέραιοι στο μέσο της λίστας ομαδοποιούνται σε μια υπο-λίστα, η οποία περικλείεται από τα επόμενα δύο μεσαία στοιχεία σε μια άλλη υπο-λίστα, η οποία περικλείεται από τα επόμενα δύο στοιχεία, κ.λπ. μέχρις ότου όλα τα στοιχεία να συμπεριληφθούν. Παραδείγματα δίνονται παρακάτω:

```
?- middle([], M).
M = [].

?- middle([1], M).
M = [1].

?- middle([1,2], M).
M = [1, 2].

?- middle([1,2,3], M).
M = [1, [2], 3].

?- middle([1,2,3,4], M).
M = [1, [2, 3], 4].

?- middle([1,2,3,4,5], M).
M = [1, [2, [3], 4], 5].

?- middle([1,2,3,4,5,6], M).
M = [1, [2, [3, 4], 5], 6].

?- middle([1,2,3,4,5,6,7], M).
M = [1, [2, [3, [4], 5], 6], 7].
```

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

```
middle([],[]).
middle([X],[X]).
middle([X,Y],[X,Y]).
middle([First,Rest],[First,Middle,Last]):-
    append(Inside,[Last],Rest),
    Inside=[_|_],          % εδώ!
    middle(Inside,Middle).

/* Όχι ντετερμινιστικό ~> -0,1
   Αν ξεχάσεις τη γραμμή με το
   "εδώ!" ~> -0,2
   γιατί αν το Inside=[] βγάζει
   και εφαλμένες απαντήσεις, πχ για [1,2] */
```

Καλή επιτυχία!