# **Εξέταση Γλώσσες Προγραμματισμού**

Θοδωρής Φρίξος Παπαρρηγόπουλος
el18040

## **Θέμα 1**

a) <S> -> ( <L> ) -> ( <L> , <S> ) -> ( <L>, <S>,  <L>) -> (<S>, a, <S>) ->
((<L>), a, (<L>)) - > ((<L>,<S>), a, (a)) -> ((<S>,a),a,(a)) -> ((a,a),a,(a)).

b)
Η γραμματική δεν είναι ambiguousκαθώς η γραμματική αυτή είναι αριστερά προσεταιριστική και ορίζει την προτεραιότητα στα “,”, και “()”.

c)  Η γραμματική παράγει είτε σκέτο α είτε tuples με τερματικό μόνο το α διαχωρισμένα με κόμα. Δηλάδη (a,a,a) ή (a,(a), (a,a,a))

## **Θέμα 2**

a.
```
fun common.. x y =
        let
                fun aux(h1::t) (h2::t2) prefix =
                        if h1 = h2 then aux t1 t2 (prefix @ [h1])
                        else (prefix, (h1::t), (h2::t2)
                | aux s1 s2 prefix = (prefix, s1,s2)
        in aux x y []
        end;
```

b. Αν του περάσουμε για λίστα το [1,1,2,3,4,5] τότε θα αληθεύσει. Θα failari το 2ο unique όμως θα μπει στο 3ο.

```
unique([]).
unique([Item | Rest]):-
   \+ member(Item, Rest), unique(Rest).
```

c.
Το ΑΜ είανι 040 -> AM1 = AM3 = 0 και ΑΜ2 = 4
γ1. 4 17 0 42 4 17
γ2. 4 17 0 42 42

d. Το ΑΜ είανι 040 -> AM1 = AM3 = 0 και ΑΜ2 = 4
δ1. 4 0 4 0
δ2. 4 4 0 0

f(4) - > g(4,0) -> print (4,0) -> x = 0 -> print(4) -> print(0)

f(4) -> g(4,0)-> print(4, 4) -> x_f = 0 -> print(x_f) = 0 -> print(x) = 0

**Θέμα 4**

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

fun trim Leaf = [Leaf]
  | trim Node(value, left, right) =
  let
    fun is_different(value, Leaf) = false
      | is_different(value, Node(v,l,r)) =
        if value mod 2 = 1 then true
        else false

    fun help(Leaf, acc) = acc
      | help(Node(n, l, r), acc) =
        let
          val left = is_different(n, l)
          val right = is_different(n,r)

        in
        end


  in
    (help(tree, []))
  end;
```

**Θέμα 5**

a)
n(_,_,_).

```
find_max(n(A,B,C), Res):- integer(A),integer(B),integer(C), M1 is max(A,B), Res is max(M1,C).
find_max(n(A,B,C), Res):- integer(A), integer(B), M1 is max(A,B), find_max(C, M2), Res is max(M1,M2).
find_max(n(A,B,C), Res):- integer(A), integer(C),  M1 is max(A,C), find_max(B, M2), Res is max(M1,M2).
find_max(n(A,B,C), Res):- integer(B), integer(C),  M1 is max(B,C), find_max(A, M2), Res is max(M1,M2).

find_max(n(A,B,C), Res):- integer(A), find_max(B,M1), find_max(C,M2), M3 is max(M1,M2), Res is max(A,M3).
find_max(n(A,B,C), Res):- integer(B), find_max(A,M1), find_max(C,M2), M3 is max(M1,M2), Res is max(B,M3).
find_max(n(A,B,C), Res):- integer(C), find_max(A,M1), find_max(B,M2), M3 is max(M1,M2), Res is max(C,M3).
find_max(n(A,B,C), Res):- find_max(A,M1), find_max(B,M2), find_max(C,M3), M4 is max(M1,M2), Res is max(M4, M3).


maximize(n(A,B,C), MaxTree):-
   find_max(n(A,B,C), Max),
   updateTree(n(A,B,C), MaxTree, Max).

updateTree(n(A,B,C), MaxTree, Max):- integer(A),integer(B),integer(C), MaxTree = n(Max,Max,Max).
updateTree(n(A,B,C), MaxTree, Max):- integer(A),integer(C), updateTree(B, T, Max), MaxTree = n(Max, T ,Max).
updateTree(n(A,B,C), MaxTree, Max):- integer(A),integer(B), updateTree(C, T, Max), MaxTree = n(Max, Max, T).
updateTree(n(A,B,C), MaxTree, Max):- integer(C),integer(B), updateTree(A, T, Max), MaxTree = n(T, Max ,Max).
updateTree(n(A,B,C), MaxTree, Max):- integer(A), updateTree(B,T1,Max), updateTree(C,T2,Max), MaxTree = n(Max,T1,T2).
updateTree(n(A,B,C), MaxTree, Max):- integer(B), updateTree(A,T1,Max), updateTree(C,T2,Max), MaxTree = n(T1,Max,T2).
updateTree(n(A,B,C), MaxTree, Max):- integer(C), updateTree(B,T1,Max), updateTree(A,T2,Max), MaxTree = n(T2,T1,Max).
updateTree(n(A,B,C), MaxTree, Max):- updateTree(A,T,Max), updateTree(B,T1,Max), updateTree(C,T2,Max), MaxTree = n(T,T1,T2).
```

b)

```
n(_,_,_).
is_odd_sum(n(A,B,C)):- integer(A),integer(B),integer(C), Sum is A + B + C, Sum mod 2 =:= 1.

unoddsum(n(A,B,C), Term):- integer(A),integer(B),integer(C),
(
   is_odd_sum(n(A,B,C)) -> Term is 17;
   Term = n(A,B,C)
).
unoddsum(n(A,B,C), Term):- integer(A),integer(B), unoddsum(C, T1),
(
   integer(T1), is_odd_sum(T1) ->
      (
         is_odd_sum(n(A,B,17))-> Term is 17;
         Term = n(A,B,17)
      );
   Term = n(A,B,T1)
).
unoddsum(n(A,B,C), Term):- integer(A),integer(C), unoddsum(B, T1),
(
   integer(T1), is_odd_sum(T1) ->
      (
         is_odd_sum(n(A,17,C))-> Term is 17;
         Term = n(A,17,C)
      );
   Term = n(A,T1,C)
).
unoddsum(n(A,B,C), Term):- integer(B),integer(C), unoddsum(A, T1),
(
   integer(T1), is_odd_sum(T1) ->
      (
         is_odd_sum(n(17,B,C))-> Term is 17;
         Term = n(17,B,c)
      );
   Term = n(T1,B,C)
).

unoddsum(n(A,B,C), Term):- integer(A), unoddsum(B, T1), unoddsum(C,T2). % check for 17
solutions and decide
unoddsum(n(A,B,C), Term):- integer(B), unoddsum(A, T1), unoddsum(C,T2),
unoddsum(n(T1,B,T2) Term). % check for 17 solutions and decide
unoddsum(n(A,B,C), Term):- integer(C), unoddsum(B, T2), unoddsum(A,T1),
unoddsum(n(T1,T2,C) Term). % check for 17 solutions and decide
unoddsum(n(A,B,C), Term):- unoddsum(A, T1), unoddsum(B,T2), unoddsum(C,T3),
unoddsum(n(T1,T2,T3) Term). % check for 17 solutions and decide
```

c) Ναι μπορούμε! Εστω συνάρτηση που επιστρεφει το αποτέλεσμα. Βελτιώνουμε με αυτή τα υποδέντρα και έπειτα να ανακατασκευάσουμε το δέντρο μας.

**Θέμα 6**

```
def sliding(list, K):
    sums = dict()

    sum = 0
    for i in range(K):
        sum += list[i]
    sums[sum]= 1

    for i in range(K, len(list)):
        sum += list[i] - list[i - K]
        if sum in sums:
            sums[sum] += 1
        else:
            sums[sum] = 1
    ans = -1
    max_sum = 0
    for a in sums:
        if sums[a] > ans:
            ans = sums[a]
            max_sum = a
        elif sums[a] == ans:
            if max_sum < a:
                ans = sums[a]
                max_sum = a

    print(max_sum, ans)


sliding([1,4,2,3,2,1,3,4,2],4)
sliding([1, 4, 2, 3, 2, 1, 3, 4, 2], 3)
```