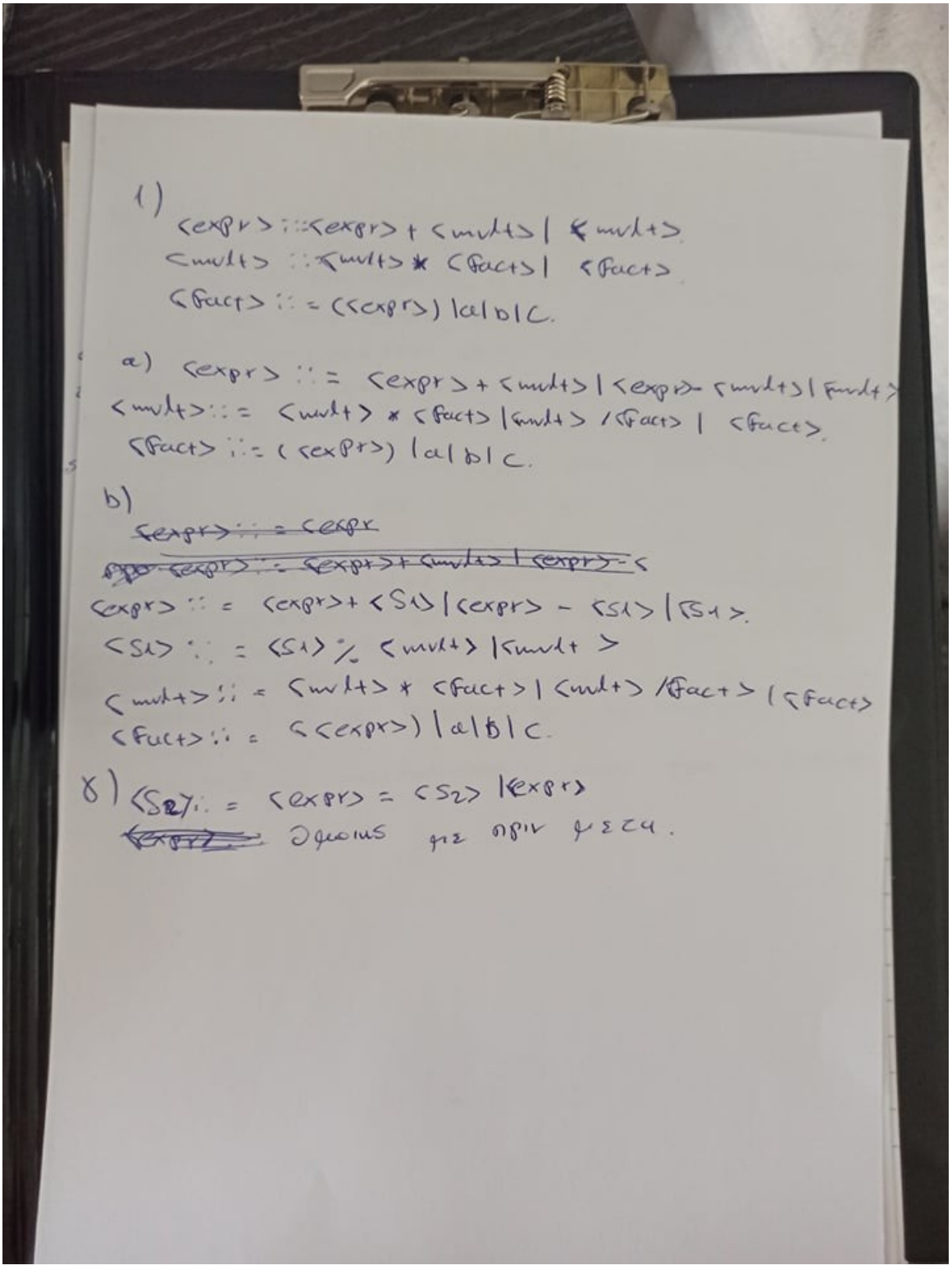
Μάνος Αραπίδης
el16071
8ο εξάμηνο

1)



3)

3)

Στην by-reference οποιαδήποτε αλλαγή κάνουμε στην στην τυπική παράμετρο ισοδυναμεί με την ίδια αλλαγή στην κανονική παράμετρο.

Στην by value-result, αρχικά η κανονική παράμετρος αντιγράφεται στην τυπική και στην επιστροφή της συνάρτησης έχουμε την αντιγραφή της τυπικής στην κανονική. Ενδιάμεσες αλλαγές στην τυπική δεν επηρεάζουν την κανονική.

Ώστε για να μην φαίνονται οι αλλαγές στην κανονική παράμετρο λόγω της εκτέλεσης του <body>, θα πρέπει αρχικά να αντιγράψουμε την τυπική παράμετρο σε μια τοπική της συνάρτησης και τέλος στον <body> θα πρέπει να αντιγράψουμε στην τυπική την τοπική

```
procedure f ( by reference x , y: integer);
    temp = x
    <body> -> αντικατάσταση όπου x -> temp

    x = temp
```

s

2)

a)

```
def f(a):
	a = list(a)
	s = sum(a)
	return [x / s for x in a]
```

b)

```
fun check k n = k * k > n orelse n mod k <> 0 andalso check (k+2) n

fun prime 2 = true
	| prime n = n mod 2 <> 0 andalso check 3 n
```

γ)

γ1)
7 17 42 1 7 17
γ2)
7 17 42 1 42 1

δ)

δ1)7 1 1 17
δ2) 1 1 17 7

5)

α)

```
max_data(n(A, L), M) :-
	max_data(L, M1),
	(M1 > A -> M is M1; M is A).

max_data([], 0).
max_data([T|Ts],M) :-
	max_data(T,MT),
	max_data(Ts,MTs),
	(MT > MTs -> M is MT; M is MTs).
```

4)

```
fun reverse ls =
	let
		fun help (nil,xs) = xs
		| help (y::ys,xs) = help (ys,y::xs)
	in help (ls,nil)
end


fun reconstruct [] = []
	| reconstruct [x] = [[x]]
	| reconstruct (z::zs) =
```

```
        let fun walk ([], _, _, acc) = acc
        (*| walk ((x::xs), sofar, i, ([]::ys)) = walk (xs, sofar, (i+1), [x]::ys)*)
        | walk ((x::xs), sofar, i, ((y::ys)::acc)) =
                if sofar > i then walk (xs, sofar, (i+1), (x::y::ys)::acc)
                else walk (xs, x, 0, [x]::(y::ys)::acc))
        in reverse (map reverse (walk (zs, z, 0, [[z]])))
end
```

6)

```
import numpy as np

def adjacency_list_to_incidence_matrix(adj_list):
        node_count = len(adj_list)
        edge_count = sum(map(len, adj_list))
        dimensions_of_matrix = (node_count,node_count)
        inc_matrix = np.zeros(dimensions_of_matrix)
        for node_index in range(0, node_count):
                for neighbor_index in adj_list[node_index]:
                        inc_matrix[node_index][neighbor_index] = 1
        return inc_matrix


def out_degree(M, u):
        l = len(M)
        count = 0
        for i in range (0,l):
                if M[u][i]==1:
                        count=count+1
        return count

def in_degree(M, u):
        l = len(M)
        count = 0
        for i in range (0,l):
                if M[i][u]==1:
                        count=count+1
        return count
```