Lucrare de laborator L1

Codul laboratorului: L1

Descriere: Programare recursiva in Lisp (1)

Data: 24.11.2020

Problema 7

a) Sa se scrie o functie care testeaza daca o lista este liniara.

$$liniara(l_1 l_2 \dots l_n) = \begin{cases} true, & l\ vida \\ false, & l\ atom \\ liniara(l_2 \dots l_n), & l_1\ atom \\ false, & l_1\ lista \end{cases}$$

b) Definiti o functie care substituie prima aparitie a unui element intr-o lista data.

$$apare(l_1 l_2 \dots l_n, e) = \begin{cases} false, & l\ vida \\ true, & l_1 = e \\ apare(l_1, e) \vee apare(l_2 \dots l_n, e), & l_1\ lista \\ apare(l_2 \dots l_n, e), & altfel \end{cases}$$

$$inlocuiesteAux(l_1 l_2 \dots l_n, target, e, ok)$$

$$= \begin{cases} \emptyset, & l\ vida \\ l_1 \oplus inlocuiesteAux(l_2 \dots l_n, target, e, ok), & ok = true \vee (l_1\ atom \wedge l_1 \neq target) \\ e \oplus inlocuiesteAux(l_2 \dots l_n, target, e, true), & l_1\ atom \wedge l_1 = target \\ inlocuiesteAux(l_1, target, e, ok) \oplus inlocuiesteAux(l_2 \dots l_n, target, e, & \\ apare(l_1, target)), & altfel \end{cases}$$

$$inlocuieste(l_1 \dots l_n, target, e) = inlocuiesteAux(l_1 \dots l_n, target, e, false)$$

c) Sa se inlocuiasca fiecare sublista a unei liste cu ultimul ei element. Prin sublista se intelege element de pe primul nivel, care este lista.
Exemplu: (a (b c) (d (e (f)))) ==> (a c (e (f))) ==> (a c (f)) ==> (a c f)
(a (b c) (d ((e) f))) ==> (a c ((e) f)) ==> (a c f)

$$ultim(l_1 l_2 \dots l_n) = \begin{cases} l_1, & n = 1 \wedge l_1\ atom \\ ultim(l_1), & n = 1 \wedge l_1 lista \\ ultim(l_2 \dots l_n), & altfel \end{cases}$$

$$inlocuiesteUltim(l_1 \dots l_n) = \begin{cases} \emptyset, & l\ vida \\ l, & l\ atom \\ l_1 \oplus inlocuiesteUltim(l_2 \dots l_n), & l_1\ atom \\ ultim(l_1) \oplus inlocuiesteUltim(l_2 \dots l_n), & altfel \end{cases}$$

d) Definiti o functie care interclaseaza fara pastrarea dublurilor doua liste liniare sortate.

$$mergeAux(l_1 l_2 \dots l_n, k_1 k_2 \dots k_m, ultim) = \begin{cases} \emptyset, & l\ vida \wedge k\ vida \\ k_1 \oplus mergeAux(l_1 l_2 \dots l_n, k_2 \dots k_m, k_1), & l\ vida \wedge k_1 \neq ultim \\ mergeAux(l_1 l_2 \dots l_n, k_2 \dots k_m, ultim), & l\ vida \wedge k_1 = ultim \\ l_1 \oplus mergeAux(l_2 \dots l_n, k_1 k_2 \dots k_m, l_1), & k\ vida \wedge l_1 \neq ultim \\ mergeAux(l_2 \dots l_n, k_1 k_2 \dots k_m, ultim), & k\ vida \wedge l_1 = ultim \\ l_1 \oplus mergeAux(l_2 \dots l_n, k_1 k_2 \dots k_m, l_1), & l_1 \leq k_1 \wedge l_1 \neq ultim \\ mergeAux(l_2 \dots l_n, k_1 k_2 \dots k_m, ultim), & l_1 \leq k_1 \wedge l_1 = ultim \\ k_1 \oplus mergeAux(l_1 l_2 \dots l_n, k_2 \dots k_m, k_1), & k_1 < l_1 \wedge k_1 \neq ultim \\ mergeAux(l_1 l_2 \dots l_n, k_2 \dots k_m, ultim), & k_1 < l_1 \wedge k_1 = ultim \end{cases}$$

$$merge(l_1 l_2 \dots l_n, k_1 k_2 \dots k_m) = \begin{cases} \emptyset, & l\ vida \wedge k\ vida \\ k_1 k_2 \dots k_m, & l\ vida \\ l_1 l_2 \dots l_m, & k\ vida \\ l_1 \oplus mergeAux(l_2 \dots l_n, k_1 k_2 \dots k_m, l_1), & l_1 \leq k_1 \\ k_1 \oplus mergeAux(l_1 l_2 \dots l_n, k_2 \dots k_m, k_1), & altfel \end{cases}$$