Examen PP/2CB/14.06.23 Timp: 2h Ex 1-9: câte 10p, Pb 10: 30p, pentru maxim sunt suficiente 100p/120p

Numele și grupa	1	2	3	4	5	6	7	8	9	10

1. În Calculul Lambda se definesc:

```
add = λm.λn.λf.λx.((m f) ((n f) x))
two = λf.λx.(f (f x))
```

Efectuați primele 3 beta-reduceri din calculul ((add two) two), folosind reducere stânga-dreapta (puteți păstra notația `two` acolo unde nu este necesar să explicitați acest termen).

2. Știind că or din Racket este o funcție nestrictă care (pe cazul de or binar) funcționează după următoarele axiome:

```
or(true, y) = true
or(false, y) = y,
```

argumentați pe scurt ce fel de recursivitate (pe stivă/pe coadă) vor genera apelurile (f '(2 3)), respectiv (g '(2 3)). **(8p)**
De asemenea, precizați la ce se evaluează aceste apeluri. **(2p)**

```
(define (f L)
  (or (f (map add1 (reverse L)))
      (odd? (car L))))
```

```
(define (g L)
  (or (odd? (car L))
      (g (map add1 (reverse L)))))
```

3. a) În Racket, **fără a folosi recursivitate explicită**, implementați funcția **insert-everywhere** care primește un element x și o listă L și îl inserează pe x în toate pozițiile în lista L, întorcând lista rezultatelor. **(6p)**
Ex: pentru x=3 și L = (1 2), rezultatul este ((3 1 2) (1 3 2) (1 2 3))
Funcții utile: (take L nr) – întoarce primele nr elemente din L
(drop L nr) – întoarce L fără primele nr elemente
(range nr) – întoarce lista (0 .. nr-1) (ex: (range 5) întoarce (0 1 2 3 4))

b) Tot **fără a folosi recursivitate explicită** și considerând implementată funcția de la punctul a, implementați funcția **insert-in-all-lists** care primește un element x și o listă de liste LL și îl inserează pe x în toate pozițiile în fiecare listă din LL, întorcând lista rezultatelor. **(4p)**
Ex: pentru x=3 și LL = ((2 1) (1 2)), rezultatul este ((3 2 1) (2 3 1) (2 1 3) (3 1 2) (1 3 2) (1 2 3))

4. Folosind funcția **insert-in-all-lists** (puteți presupune că ea există, chiar dacă nu ați implementat-o), implementați în Racket fluxul infinit (((1)) ((2 1) (1 2)) ((3 2 1) (2 3 1) (2 1 3) (3 1 2) (1 3 2) (1 2 3)) …), în care elementul de pe poziția i reprezintă toate permutările listei (1 .. i), fără să conteze ordinea în care apar aceste permutări.

5. Cunoscând div, mod :: Integral a => a -> a -> a (și apartenența la clasa Integral este singura constrângere legată de operațiile cu tipuri numerice), sintetizați tipul lui lst (nu sunt necesare explicații formale, dar sunt necesare explicații).

```
lst lim = [ (n, if null ds then Nothing else Just ds ) |
            n <- [2 .. lim], let ds = [ d | d <- [2 .. div n 2], mod n d == 0 ] ]
```

6. Fie tipul Tournament care descrie un turneu de tenis prin nume și lista de puncte pe care turneul le acordă pentru victoria în diverse runde (ex: T “Roma” [20, 25, 45, 90, 180, 240, 400] înseamnă că la turneul de la Roma, cine câștigă în prima rundă primește 20 de puncte, la care se adaugă 25 pentru victorie în runda 2, 45 pentru victorie în runda 3, etc.):

```
type Name = String
type Points = Int
data Tournament = T Name [Points] deriving Show
```

Adăugați tipul Tournament la clasa Eq, știind că 2 turnee sunt egale dacă oferă același număr total de puncte învingătorului turneului și aceste puncte se obțin după câștigarea aceluiași număr de meciuri.

7. Traduceți în FOL (first order logic) propoziția: „Există un animal mai mic decât toți câinii cărora le place să înoate”.

8. Implementați, în Prolog, predicatele **prefix**(+A, +B) **(5p)** și **sufix**(+A, +B) **(5p)**, care primesc 2 liste **nevide** A și B și verifică dacă A este un prefix, respectiv sufix al listei B. (ex: prefix([1,2], [1,3,2,5]) va fi false, sufix([2,4], [0,2,4]) va fi true)

9. Considerând implementate predicatele prefix și sufix de mai sus, definiți într-**o singură regulă** predicatul **can_decompose**(+A, +B) care primește 2 liste **nevide** A și B și verifică dacă A poate fi scris ca o concatenare de liste **nevide** X și Y astfel încât X este un prefix și Y este un sufix pentru B. Corpul regulii can_decompose trebuie să constea într-**un singur metapredicat** (satisfacerea acestuia generează răspunsul true, nesatisfacerea sa generează false).
Ex: can_decompose([1,1,3],[1,3]) este true, pentru că [1] este prefix pentru [1,3], iar [1,3] este sufix pentru [1,3]

Examen PP/2CB/14.06.23 Timp: 2h Ex 1-9: câte 10p, Pb 10: 30p, pentru maxim sunt suficiente 100p/120p

10. Adăugăm următoarele definiții tipului Tournament de la exercițiul 6:

```
-- set = (game-uri câștigate, game-uri pierdute) - din perspectiva jucătorului curent
type Set = (Int, Int)
-- scor = listă de 2 sau 3 seturi (câștigă cine câștigă 2 seturi)
type Score = [Set]
-- performanța unui jucător la turneul cu numele name = listă de scoruri
-- (atâtea scoruri câte runde a jucat până a pierdut sau a câștigat turneul)
data TournamentRecord = TR Name [Score]
-- un jucător e descris prin numele său și o listă de performanțe la diverse turnee
data Player = P Name [TournamentRecord]
```

În rezolvarea fiecărei cerințe de mai jos este obligatoriu să folosiți **ori o funcțională, ori un list comprehension**:

a) Implementați funcția **playerWins :: Score -> Bool** care primește un scor și întoarce True dacă jucătorul (al cărui punctaj apare pe prima poziție în rezultatul fiecărui set) a câștigat acel meci, și False în caz contrar. **(10p)**
Ex: playerWins [(6,2), (6,7), (6,7)] => False

b) Implementați funcția **findTournamentByName :: Name -> [Tournament] -> Tournament**, care primește numele unui turneu și o listă de turnee și întoarce turneul cu numele respectiv (care se află, garantat, în listă). **(10p)**
Ex: findTournamentByName "Madrid" [T "Roma" [1,2,3,4,5], T "Madrid" [2,4,6,8,10]] => T "Madrid" [2,4,6,8,10])

c) Implementați funcția **playerPointsInTournament :: Player -> Tournament -> Points**, care primește un jucător și un turneu și calculează totalul punctelor acumulate de jucător în turneul respectiv. **(10p)**
Ex: pentru un jucător în al cărui TournamentRecord apare valoarea TR “Roma” [[(6,1), (6,1)], [(4,6), (7,6), (7,5)], [(3,6), (4,6)]] și turneul T "Roma" [1,2,3,4,5], funcția întoarce 1+2 = 3, pentru că jucătorul a câștigat primele 2 runde (acumulând punctele acordate pentru câștigarea lor), apoi a pierdut