# Examen PP – Seria CD — NOT EXAM MODE

14.06.2017

Timp de lucru 2 ore . 100p necesare pentru nota maximă

---

1. Determinați forma normală pentru următoarea expresie, ilustrând pașii de reducere:
   $((\lambda x.\lambda y.\lambda z.y\ \ \Omega)\ \ \lambda x.(x\ \ z))$
   *Soluție:*
   $((\lambda \underline{x}.\lambda y.\lambda z.y\ \ \Omega)\ \ \lambda x.(x\ \ z)) \rightarrow_\beta (\lambda \underline{y}.\lambda z.\underline{y}\ \ \lambda x.(x\ \ z)) \rightarrow_\alpha (\lambda \underline{y}.\lambda w.\underline{y}\ \ \lambda x.(x\ \ z)) \rightarrow_\beta \lambda w.\lambda x.(x\ \ z)$

2. Ce întoarce următorul cod Racket și ce reprezintă valoarea întoarsă?
   ```
   (let f ((n 0) (m 5)) (if (< n m) (+ n (f (+ n 1) m)) 0))
   ```
   *Soluție:*
   10, adună numerele de la n la m (fără m).

3. De câte ori se evaluează expresia `(E 1)` în codul Racket de mai jos?
   ```
   (let ((proc (λ (x) (+ (force x) (force x))))) (proc (delay (E 1))))
   ```
   Tranformați codul dat pentru a folosi închideri.
   Este vreo diferență în execuție?
   *Soluție:*
   o singură dată; `(let ((proc (λ (x) (+ (x) (x))))) (proc (λ() (E 1))))`
   se va evalua E de 2 ori.

4. Implementați în Haskell o funcție care primește 3 liste infinite și le adună element cu element. E.g. `f [1..] [4..] [7..] = [12, 15..]`
   *Soluție:*
   ```
   f = zipWith3 (\x y z -> x + y + z) sau
   f (h1:t1) (h2:t2) (h3:t3) = h1 + h2 + h3 :  f t1 t2 t3
   ```

5. Se dă clasa Haskell: `class Summable t where sum :: (Num a) => t a -> a`,
   cu semnificația că funcția din clasă adună toate elementele conținute în argument.
   - Instanțiați clasa dată pentru tipul listă;
   - Definiți un tip *listă imbricată de elemente de tip* `a`;
   - Instanțiați clasa dată pentru tipul definit.

   *Soluție:*
   ```
   instance Summable [] where sum l = foldl (+) 0 l
   data SL a = A a | S [SL a] deriving (Eq, Show)
   instance Summable SL where sum a = case a of (A a) -> a; (S l) -> foldl (+)
   0 $ map sum l
   ```

6. Știind că *Vulpea care doarme nu prinde găini* , și că $vulpe(Vicky)$, $gaina(Gini)$, și $doarme(Vicky)$ , demonstrați folosind rezoluția că $prinde(Vicky, Gini)$ este fals .
   *Soluție:*
   $\forall x.\forall y.vulpe(x) \wedge gaina(y) \wedge doarme(x) \rightarrow \neg prinde(x, y)$
   *sau* $\forall x.vulpe(x) \wedge doarme(x) \rightarrow \neg\exists y.gaina(y) \wedge prinde(x, y)$

   $\neg vulpe(x) \vee \neg gaina(y) \vee \neg doarme(x) \vee prinde(x, y)$

$+vulpe(Vicky)\{x \leftarrow Vicky\} \Rightarrow \neg gaina(y) \vee \neg doarme(Vicky) \vee prinde(Vicky, y)$
$+\neg prinde(Vicky, Gini)y \leftarrow Gini\} \Rightarrow \neg gaina(Gini) \vee \neg doarme(Vicky)$
$+doarme(Vicky) \Rightarrow \neg gaina(Gini)$
$+gaina(Gini) \Rightarrow$ clauza vidă

7. Implementați în Prolog un predicatul `p(L1, L2, L3)`, care primește în L1 și L2 două liste de lungimi egale și leagă L3 la o listă de perechi, fiecare pereche conținând elementele de la același index din L1 și L2 (similar `zip` din Haskell) . Exemplu: este adevărat `p([1, 2, 3], [a, b, c], [(1, a), (2, b), (3, c)]).`
   *Soluție:*
   ```
   zip(A, B, C) :- findall((X, Y), (nth0(I, A, X), nth0(I, B, Y)), C).
   ```

8. Implementați un algoritm Markov care primește un șir format din simboluri `0` și `1` și verifică dacă șirul începe și se termină cu același simbol. Dacă simbolurile de la început și de la sfârșit sunt identice, scrie la sfârșitul șirului simbolurile "`ok`", altfel nu schimbă șirul cu nimic. Încercați să scrieți algoritmul pentru un alfabet de bază A oarecare.
   *Soluție:*
   1. `StartEnd(); A` $g_1$, $g_2$
   2. $ag_1 \rightarrow g_1 bg_1$
   3. $bg_1 g_2 \rightarrow g_2 bg_1$
   4. $g_1 bg_1 \rightarrow g_1 okb$
   5. $g_1 bg_2 \rightarrow g_1 b$
   6. $b \rightarrow .$
   7. $\rightarrow a$

A