Nume: _____

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 10 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |

**Examen PP CC - var C** 21.06.2024

ATENȚIE: Aveți 2 ore · 1-9: 10p; 10: 30p · 100p pentru nota maximă · **Justificați** răspunsurile!

---

1. Pentru următoarea $\lambda$-expresie: (a) câți beta-redecși există în expresie? (b) ce variabile *distincte* există în expresie?: $((\lambda x.x\ \ y)\ \ \lambda x.(x\ \ x))$

2. Scrieți în Racket funcția `ascendingKeys` care primește o listă de perechi cheie-valoare și întoarce adevărat dacă pentru fiecare pereche fie este singura pereche cu acea cheie, fie următoarea pereche cu aceeași cheie are valoarea strict mai mare. De exemplu, pentru lista `((a . 1) (b . 2) (a . 3) (b . 4))` funcția întoarce adevărat, dar pentru `((a . 1) (b . 2) (c . 3) (a . 0) (b . 4))` întoarce fals pentru că valorile pentru cheia `a` sunt 1 și apoi 0. În implementare folosiți **cel puțin o funcție explicit recursivă, ce folosește recursivitate pe coadă**. Indicați care este acea funcție.

3. Implementați funcția de la exercițiul anterior **fără a folosi recursivitate explicită**.

4. Construiți în Racket sau în Haskell, folosind o definiție **implicită**, fluxul [0,1,1,2,4,7,13,24,44,81] în care fiecare element în afară de primele 3 este suma ultimelor 3 elemente.

5. Sintetizați, **ilustrând** procesul de sinteză, tipul funcției Haskell `f x = filter (\a -> a x)`

6. Instanțiați clasa Num (doar operatorii de adunare și înmulțire) pentru tipul listă Haskell, unde 'adunarea' a două liste rezultă în adunarea lor element cu element, iar 'înmulțirea' a două liste rezultă în selectarea celei mai scurte dintre liste.

7. Știind că „Nu aduce anul ce aduce ceasul" (folosind predicatul $aduce(Cine, Ce)$) și știind că $aduce(ceasul, noroc)$, demonstrați că $\neg aduce(anul, noroc)$.

8. Implementați în Prolog predicatul `remove(X, List, Result)` care produce în `Result` o listă identică cu `List`, cu excepția că în `Result` nu apare nicio apariție a elementului `X`. Implementați folosind recursivitate explicită.

9. Implementați predicatul de la exercițiul anterior folosind **metapredicate**.

10. Se consideră un tip de date, pe care îl numim *serie de valori*, care este o enumerare de înregistrări, fiecare înregistrare având o cheie și o valoare. Cheile dintr-o serie de valori sunt toate de același tip `t`. Valorile sunt toate numere întregi. De exemplu, putem avea seria de valori a:1, a:2, b:10, c:10, b:9, c:5, a:3.
    (a) definiți în Haskell tipul descris mai sus. Implementați funcția `selectSeries` care, pentru o cheie de tip `t` și o serie de valori peste tipul `t`, întoarce o listă formată din valorile asociate cu acea cheie, în ordinea în care au fost înregistrate. Pentru exemplu și cheia `a`, funcția întoarce lista `[1,2,3]`.
    (b) implementați funcția `valueCounts` care pentru o serie de valori întoarce o listă de perechi unde primul element este o cheie din serie, iar al doilea element este numărul de valori asociate cu acea cheie. Pentru exemplul de mai sus, funcția întoarce `[('a',3),('b',2),('c',2)]`.
    (c) implementați funcția `allMonotonic`, care pentru o serie de valori verifică că evoluția este monoton crescătoare sau descrescătoare *pentru fiecare cheie în parte*. Pentru exemplu, funcția întoarce `True`.