Nume: _____

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

Examen PP CC - var B 21.06.2024

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

---

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

2. Scrieți în Racket o funcția `serialChecker` care primește o listă de perechi, fiecare pereche având pe prima poziție o funcție și pe a doua poziție o listă. `serialChecker` verifică pentru fiecare pereche dacă cel puțin unul dintre elementele listei este validat de funcția din pereche. Exemplu: `(serialChecker (list (cons odd? '(1 3 5)) (cons even? '(2 3 4)) (cons null? '(() () ())) (cons zero? '(1 2 3))))` → `'(#t #t #t #f)` 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 1+0, 1-1, 0+2, 2-3, -1+4, 3-5, . . . , adică 1, 0, 2, -1, 3, -2, . . . , în care fiecare element este elementul anterior din care se adună sau scade, alternativ, indexul elementului în flux.

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

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

7. Știind că „Nu este pentru cine se potrivește, ci este pentru cine se nimerește", și știind că $se\_nimerește(Marcel, Irina)$, demonstrați folosind metoda rezoluției că $este\_pentru(Marcel, Irina)$.

8. Implementați în Prolog predicatul `select(List, X, 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`.