Examen PP/06.06.25 Exercițiile 1-9: câte 10p, Problema 10: 30p Nume / Grupă:
Timp: 2h Pentru maxim sunt suficiente 100p/120p

A

1	2	3	4	5	6	7	8	9	10a	10b	10c

Calcul Lambda

1. a) Scrieți o λ-expresie care conține fix 2 variabile (x și y), cu 2 apariții fiecare, astfel încât 3 dintre apariții să fie libere și una singură legată.
   b) Încercuiți aparițiile legate.

Racket

2. Implementați recursiv funcția num-to-digit care adună în mod repetat cifrele unui număr până când rămâne cu o singură cifră.
   - ex: `2587 -> 2+5+8+7 = 22 -> 2+2 =` 4 (rezultatul apelului este 4)
3. a) Implementați funcția f care numără de câte ori este adevărat un predicat p între 2 elemente consecutive ale unei liste date L.
   - ex: `p = equal?, L = '(1 1 2 2 2) ->` 3 `grupuri: '(1 1), '(2 2), '(2 2)`
   - **Constrângeri:** - f trebuie să fie curry într-un mod util punctului b)
     - folosiți funcționale, nu folosiți recursivitate explicită

   b) Implementați funcția f1 care determină câte elemente ale listei argument sunt mai mici decât jumătate din predecesorul lor.
   - **Constrângeri:** - f1 trebuie să fie o aplicație parțială a lui f
     - folosiți funcții anonime
4. Implementați funcția s care primește o listă L_0 și întoarce fluxul L_0, L_1, L_2, ... unde L_i se obține din L_{i-1} prin incrementarea elementelor de pe poziții divizibile cu 3. Considerați lista indexată de la 0.
   - ex: L_0 `= '(1 1 1 1 1) ->` `'(1 1 1 1 1), '(2 1 1 2 1), '(3 1 1 3 1) …`

Haskell

5. Scrieți o funcție care are signatura (Ord a, Num b) => (a -> b) -> [a] -> [(a, b)].
6. Modelăm un joc de ghicire a unui număr target, în care fiecare tentativă primește un feedback de tip "prea mare", "prea mic", sau "la fix", folosind următoarele tipuri de date:

```
data GuessingGame = GG { target :: Int, guesses :: [Guess] } deriving Show
data Result = TooLow | TooHigh | SpotOn deriving Show
data Guess = Guess { attempt :: Int, result :: Result } deriving Show
data OrdGuess = OrdGuess { guess :: Guess, reference :: Int } deriving
                                                              Show
```

   Tipul OrdGuess reține atât o încercare guess cât și target-ul (referința) cu care a fost comparată această încercare.
   Adăugați tipul OrdGuess la clasa Ord astfel încât valorile tipului să fie ordonate după cât de apropiate sunt încercările de target (mai aproape = mai mic). Asigurați-vă că îndepliniți toate precondițiile adăugării tipului la clasa Ord.

Logică

7. Aduceți propoziția $((P \vee Q) \wedge \neg P) \Longrightarrow Q$ la forma clauzală, precizând pașii efectuați.

Prolog

8. Implementați predicatul longest_consecutive(+L, -N) care determină cel mai mare număr de elemente consecutive egale din lista L. Util: funcția max/2 (ex utilizare: M is max(2, 3)).
   - ex: `longest_consecutive([1,2,3,3,4,3,2,2,2,3,3], N) -> N = 3`
9. Fie un program care gestionează o bibliotecă, modelată prin fapte de tip carte(Titlu, Autor, An, Gen).
   - ex: `carte('Dune', 'Frank Herbert', 1965, science_fiction).`
   Implementați predicatul author_genre_pairs(-Pairs), care determină lista tuturor perechilor unice (Autor, Gen) din bibliotecă.
   - **Constrângeri:** - folosiți metapredicate, nu folosiți recursivitate explicită

Problema (Haskell)

10. Pentru jocul și tipurile descrise la exercițiul 6:
    a) (10p) Implementați funcția isWin :: GuessingGame -> (Bool, Maybe Int), care primește un joc și întoarce o pereche între:
    - un boolean (adevărat dacă jocul a fost câștigat, fals altfel)
    - numărul de încercări efectuate până la victorie - în caz că jocul a fost câștigat (se folosește un Maybe Int pentru a acoperi și cazul în care jocul nu a fost câștigat).

    - **Observație:** un joc se încheie odată cu ghicirea numărului, o încercare reușită nu va fi urmată în listă de alte încercări.

    b) (5p) Implementați funcția checkGuess :: Guess -> Maybe Int -> Maybe Int -> (Bool, Maybe Int, Maybe Int), care primește o încercare și două limite (inferioară, respectiv superioară) între care ar trebui să se afle încercarea conform istoricului de încercări, și întoarce un triplet între:
    - un boolean - adevărat dacă și numai dacă încercarea curentă a fost între limitele date
    - noua limită inferioară (actualizată conform feedback-ului primit pentru încercarea curentă)
    - noua limită superioară (actualizată conform feedback-ului primit pentru încercarea curentă)

    - **Observație:** Limitele sunt modelate cu un Maybe pentru că este nevoie de minim o încercare prea mică/mare pentru a avea o limită inferioară/superioară. Trebuie tratat și cazul în care limitele (una sau ambele) nu au fost încă setate.

    (5p) Apoi implementați funcția isPlayerLogical :: GuessingGame -> Bool, care verifică dacă șirul de încercări este logic (nu s-a făcut niciodată o încercare inutilă - în afara limitelor stabilite prin încercările anterioare).

    c) (10p) Implementați funcția bestGuess :: GuessingGame -> Maybe Guess care determină cea mai bună încercare efectuată de-a lungul jocului, adică încercarea cea mai apropiată de target (la egalitate, se poate întoarce oricare dintre opțiuni). Folosim un Maybe pentru a acoperi și situația dinaintea efectuării primei încercări.