Varianta B – Examen PP (2020)

1. `(λx y z.corp a b c)` este forma prescurtată de a scrie `(((λx.λy.λz.corp a) b) c)`.

   Fie următoarele definiții în Calcul Lambda:

   `true = λx y.x`

   `false = λx y.y`

   și fie operatorul logic:

   `op = λx y.(x true y)`

   Descrieți pas cu pas (nu efectuați mai multe β-reduceri deodată) comportamentul lui `op` pe valori de tip `true` sau/și `false`, astfel încât să puteți conchide la final că `op` se comportă ca unul dintre operatorii logici cunoscuți (și precizați care este operatorul a cărui identitate a fost ascunsă prin folosirea numelui "`op`").

2. Ce tip de recursivitate are implementarea de mai jos?

```
(define (f x)
    (let h ((x x) (r '()))
        (cond ((null? x) (reverse r))
            ((list? (car x)) (h (append (car x) (car x) (cdr x)) r))
            (else (h (cdr x) (cons (car x) r))))))
```

   Definiți (tot în Racket), eliminând nevoia folosirii unui helper, funcția `ff` care are același efect cu `f`, dar folosește un alt tip de recursivitate.

   Precizați pe scurt care modificare din cod a dus la modificarea tipului de recursivitate.

3. **Atenție:** Acest exercițiu nu se punctează decât dacă este rezolvat fără a folosi recursivitate explicită (deci nici `letrec` sau `named let`). Soluțiile explicit recursive se punctează cu 0.

   Se dă o listă `L` care conține numere și/sau liste de numere și/sau liste de numere (maxim 2 niveluri de imbricare). Să se definească, în Racket, funcția `count-nulls` care numără listele vide aflate pe orice nivel de imbricare.

   ex: `(count-nulls '(() 1 (2 () (3 4 5)) (())))` va întoarce 3, pentru că avem o listă vidă direct în `L`, și încă 2 în liste interioare lui `L` (acesta fiind și nivelul maxim de imbricare admis, deci nu trebuie căutate liste vide la adâncime mai mare, de exemplu `'(0 (1 (2 ())))` nu este un input valid pentru problemă).

4. Definiți în Racket următorul flux:

```
'((1) (2 1 2) (3 2 1 2 3) (4 3 2 1 2 3 4) (5 4 3 2 1 2 3 4 5) ...)
```

5. Sintetizați în Haskell, pas cu pas, tipul funcției `g`:

```
g f (x, y) = x ++ map f y
```

6. Scrieți o instanță posibilă a clasei de mai jos, conținând o implementare **neconstantă** a funcției `f`.

```
class MyClass c where
    f :: Ord a => c a -> c a -> c Bool
```

7. Folosiți pentru scriere `NOT()`, `AND` și `OR` ca operatori logici și `ORICARE` și `EXISTĂ` drept cuantificatori. Scrieți cu caps ca să nu existe confuzii cu limbajul obișnuit. Puteți prescurta numele predicatelor.

   Știind că "Capul plecat, sabia nu îl taie", și că `i_se_taie_capul_lui(marcel)`, dorim să demonstrăm că `NOT(are_capul_plecat(marcel))`.

   a) Traduceți proverbul în LPOI (FOL) utilizând numai predicatele menționate în problemă.

   b) Demonstrați **prin metoda rezoluției** concluzia cerută.

   Nu se punctează demonstrațiile care nu folosesc metoda rezoluției.

8. Implementați predicatul `gen(+Sample, +Length, -Output)`, care produce în `Output` o listă de lungime `Length` care constă din repetări ale listei `Sample`.

   Exemplu: `g([1, 2, 3], 10, X)` leagă `X` la `[1, 2, 3, 1, 2, 3, 1, 2, 3, 1]`.

   Explicați cum funcționează implementarea.

9. Folosiți o singură dată unul dintre predicatele `findall`, `forall`, `bagof`, `setof` pentru a implementa predicatul `minmax(+L, -Min, -Max)` care leagă `Min`, respectiv `Max`, la elementul minim, respectiv maxim, al listei. Nu utilizați recursivitate explicită. Explicați cum funcționează soluția.

10. În acest exercițiu urmărim crearea și exploatarea unei tabele de frecvență pentru caracterele care apar în cuvinte diferite dintr-un text. O tabelă de frecvență este o colecție de asocieri între un caracter și frecvența sa de apariție.

    Utilizați următorul schelet:

```
import Data.List
import Data.Char
import Data.Maybe
```

- (10p) Să se creeze aliasul de tip `FreqTable`, pentru tabele de frecvență.
  Să se implementeze funcția `rareChars`, care primește o tabelă de frecvență și un număr n și întoarce lista caracterelor din tabelă care au frecvența mai mică decât n.
  Exemplu:

```
*Main> rareChars [('a',3), ('b',1), ('c',2), ('d',1)] 2
"bd"
```

```
type FreqTable = Int -- aici trebuie modificat în altceva decât Int

rareChars :: FreqTable -> Int -> [Char]
rareChars = undefined
```

- (10p) Să se implementeze funcția `insChar`, care inserează un caracter într-o tabelă de frecvență (dacă el exista deja în tabelă, atunci frecvența este actualizată).
  Exemplu:

```
*Main> insChar 'e' [('a',3), ('b',1), ('c',2), ('d',1)]
[('e',1),('a',3),('b',1),('c',2),('d',1)]
*Main> insChar 'c' [('a',3), ('b',1), ('c',2), ('d',1)]
[('c',3),('a',3),('b',1),('d',1)] -- s-a actualizat frecvența lui c
```

```
insChar :: Char -> FreqTable -> FreqTable
insChar = undefined
```

- (10p) Să se implementeze funcțiile `textToStr` și `textToTable`.
  `textToStr` primește un text și creează un string în care fiecare cuvânt din text apare o singură dată, și sunt eliminate spațiile albe.
  `textToTable` primește un text și creează o tabelă de frecvență pentru cuvintele distincte din acesta (folosind rezultatul lui `texToStr`).
  Util:

```
words :: String -> [String] -- împarte un text în cuvinte, parsând după
↪  spații
```

  Exemplu:

```
*Main> textToStr "copacii albi copacii negri"
"copaciialbinegri"
*Main> textToTable "copacii albi copacii negri"
[('c',2),('o',1),('p',1),('a',2),('i',4),('l',1),('b',1),('n',1),('e',1),('g',1),('r',1)]
```

```
textToStr :: String -> String
textToStr = undefined

textToTable :: String -> FreqTable
textToTable = undefined
```