Precizări:

- Fiecare subiect are câte 10p.
- Este suficientă rezolvarea a 10 subiecte din cele 12 pentru nota maximă.
- Punctajul suplimentar reprezintă bonus, ce poate compensa punctajul de pe parcurs.

1. Care este rezultatul evaluării aplicației $(x \quad \lambda x.x)$, în absența oricăror informații despre variabila liberă $x$? Justificați!

   *Soluție.* .

   Evaluarea se blochează deoarece membrul stâng al aplicației nu poate fi adus la forma funcțională. □

2. Implementați în Haskell operatorul `(++)`, întrebuințând una din funcționalele **`fold*`** și **fără** a utiliza recursivitate explicită. Soluțiile care nu respectă cele două constrângeri **nu** vor fi punctate!

```
foldl :: (a ->  b  -> a)
      ->  a -> [b] -> a
foldr :: (a ->  b  -> b)
      ->  b -> [a] -> b
```

   *Soluție.* .

```
append xs ys = foldr (:) ys xs
```

   □

3. La ce se evaluează ultima expresie a programului Scheme de mai jos? Justificați!

```
(define f (lambda () x))
(let ([x 1]) (f))
```

   *Soluție.* .

   Evaluarea generează eroare, deoarece variabila `x` din corpul funcției `f` nu este găsită. Din moment ce construcția `let` realizează legare statică, domeniul de vizibilitate a lui `x`, definit aici, nu include funcția `f`. □

4. Fie funcția Scheme de mai jos:

```
(define f
  (lambda (x y)
    (if (< x 5)
        (+ x x)
        (+ y y))))
```

   Precizați cum decurge, pas cu pas, evaluarea expresiei `(f (+ 1 2) (+ 2 3))`, utilizând modelul de evaluare bazat pe **substituție** textuală.

*Soluție.* .

```
(f (+ 1 2) (+ 2 3))
→ (f 3 (+ 2 3))
→ (f 3 5)
→ (if (< 3 5) (+ 3 3) (+ 5 5))
→ (if #t (+ 3 3) (+ 5 5))
→ (+ 3 3)
→ 6
```

□

5. Fie funcția Haskell de mai jos:

```
1 f :: Int -> Int -> Int
2 f 2013 _ = 2013
3 f x    y = x + y
```

Ce puteți spune despre terminarea evaluării următoarei expresii Haskell?

```
foldr f 0 [0 ..]
```

*Soluție.* .

Se termină, deoarece, conform definiției lui `f`, dacă elementul curent al listei este `2013`, aplicația recursivă a lui `foldr` asupra restului listei nu mai este evaluată, ca urmare a strategiei lenește. □

6. Definiți în Haskell fluxul rezultat prin aplicarea succesivă a unei funcții cu tipul `a -> a` asupra unei expresii inițiale de tipul `a`. De exemplu, pentru funcția `(+ 1)` și valoarea inițială `0`, se obține fluxul `[0, 1, 2, 3, ...]`.

*Soluție.* .

```
1 apply f x = x : apply f (f x)
```

□

7. Sintetizați tipul operatorului `(.)`, având definiția:

```
(.) f g x = f (g x)
```

*Soluție.* .

```
1 (.) :: a -> b -> c -> d
2 f :: e -> h
3 g :: i -> j
4 a = e -> h
5 b = i -> j
6 c = i
7 e = j
8 d = h
9 (.) :: (j -> h) -> (i -> j) -> i -> h
```

□

8. Pentru tipul de date `Natural`, definit mai jos, supraîncărcați explicit (fără a utiliza `deriving`) funcția standard de verificare a egalității.

```
data Natural
    = Zero
    | Succ Natural
```

*Soluție.* .

```
instance Eq Natural where
    Zero     == Zero     = True
    Zero     == (Succ _) = False
    (Succ _) == Zero     = False
    (Succ m) == (Succ n) = m == n
```

□

9. Utilizați rezoluția propozițională pentru a demonstra că propoziția $p$ este o consecință logică a propoziției $p \wedge q$.

*Soluție.* .

Forma clauzală a mulțimii de propoziții, formată din ipoteză și negația concluziei, este $\{p\}, \{q\}, \{\neg p\}$. Aplicând rezoluția asupra clauzelor $\{p\}$ și $\{\neg p\}$, se obține clauza vidă. □

10. Definiți în Prolog predicatul `xor/2`, care să fie satisfăcut dacă **numai unul** din cei doi parametri este satisfăcut. **Nu** puteți folosi predicatul `not` sau operatorul `\+`.

*Soluție.* .

```
xor(X, Y) :- X, Y, !, fail.
xor(X, _) :- X, !.
xor(_, Y) :- Y.
```

□

11. Scrieți un algoritm Markov, care calculează predecesorul unui număr natural nenul. Numerele sunt reprezentate unar, în forma unei secvențe de simboluri `1`, de lungime egală cu numărul. De exemplu, pentru numărul 3, reprezentat prin secvența `111`, banda va conține la sfârșitul execuției secvența `11`.

*Soluție.* .

În programul de mai jos, `a` este o variabilă de lucru.

```
Predecesor()
    a -> .
    1 -> a
end
```

Exemplu: `111 -3-> a11 -2-> 11`. □

12. Dați trei exemple de fapte de lungimi diferite, care se potrivesc cu *pattern*-ul CLIPS `(f $? $?)`.

*Soluție.* .

`(f), (f 1), (f 1 2)` □