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 numărul **minim** de α-conversii ce trebuie realizate pentru evaluarea corectă a expresiei $(\lambda x.\lambda x.x \ \ \lambda x.x)$? Justificați!

   Soluție. .

   Niciuna, deoarece parametrul actual, $\lambda x.x$, nu conține variabile libere. □

2. Implementați în Haskell funcția `length`, 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. .

```
len = foldr (\_ acc -> acc + 1) 0
```

   □

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

```
(define x 1)
(define x (lambda () x))
(x)
```

   Soluție. .

   `#<procedure:x>`, deoarece `define` leagă dinamic variabilele, și se ia în considerare ultima legare a lui `x`, la funcția însăși. □

4. În ce condiții expresia Scheme `(eq? (force x) (force x))` s-ar putea evalua la `#f`? (5p) Dați o posibilă definiție a lui `x`. (5p)

   Soluție. .

   În condițiile în care forțarea promisiunii `x` produce efecte laterale explicite. Exemplu:

```
(define x (delay (set! x 1) 0))
```

   □

5. Fie funcția Haskell de mai jos:

```
f x y = if x < 5
        then x + x
        else 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)
→ if (1 + 2) < 5 then (1 + 2) + (1 + 2) else (2 + 3) + (2 + 3)
→ if 3 < 5 then 3 + 3 else (2 + 3) + (2 + 3)
→ if True then 3 + 3 else (2 + 3) + (2 + 3)
→ 3 + 3
→ 6
```

De remarcat că, o dată ce `(1 + 2)` este evaluat, în vederea comparației cu `5`, valoarea sa rămâne salvată și în cadrul expresiei `3 + 3`. □

6. Definiți în Haskell fluxul prefixelor unui alt flux, primit ca parametru. De exemplu, pentru fluxul `[0, 1, 2, ...]`, se obține fluxul `[[], [0], [0, 1], [0, 1, 2], ...]`.

Soluție. .

```
prefixes (x : xs) = [] : (map (x :) (prefixes xs))
```

□

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

```
($) f x = f x
```

Soluție. .

```
($) :: a -> b -> c
f :: d -> e
a = d -> e
b = d
c = e
($) :: (d -> e) -> d -> e
```

□

8. Pentru tipul de date `Natural`, definit mai jos, supraîncărcați funcția standard de reprezentare sub formă de șir de caractere, utilizând valoarea numerică aferentă. De exemplu, valorii `Succ (Succ Zero)` îi va corespunde șirul `"2"`.

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

Soluție. .

```
instance Show Natural where
    show = show . value
      where
        value Zero     = 0
        value (Succ n) = 1 + (value n)
```

□

9. Fie propoziția de mai jos, din logica propozițională, toate propozițiile din definiție fiind simple. Câte interpretări distincte admite ea? Justificați!

$$\alpha = ((p \vee q \vee r) \wedge (s \vee \neg r) \wedge (\neg s \vee q)) \Leftrightarrow (t \vee u)$$

Soluție. .

Numărul de propoziții simple distincte din definiție este 6, astfel că numărul de interpretări este $2^6 = 64$. □

10. Transcrieți în Prolog propoziția $\forall x.\neg rotund(x) \Rightarrow \neg cerc(x)$.

Soluție. .

Propoziția este echivalentă cu $\forall x.cerc(x) \Rightarrow rotund(x)$.

```
1 rotund(X) :- cerc(X).
```

□

11. Scrieți un algoritm Markov, care calculează restul împărțirii la 2 a unui număr natural. 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 simbolul `1`. Pentru numărul 2, reprezentat prin secvența `11`, banda va ajunge vidă.

Soluție. .

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

```
1 Mod2()
2     11 ->
3     -> .
4 end
```

Exemplu: `11111 -2-> 111 -2-> 1 -3-> 1.` □

12. Dați trei exemple de fapte de lungimi diferite, care se potrivesc cu *pattern*-ul CLIPS `(f ? 5 $?).`

Soluție. .

`(f 0 5), (f 0 5 1), (f 0 5 1 2)` □