Precizări:

- Fiecare subiect are câte 10p.
- Este suficientă rezolvarea a 10 subiecte din cele 12 pentru nota maximă.

1. Ce proprietăți trebuie să îndeplinească expresia F, astfel încât expresia E să fie reductibilă? Justificați!

$$E \equiv (((\lambda x.x \ \ \lambda x.x) \ \ \Omega) \ \ (F \ \ \Omega))$$
$$\Omega \equiv (\lambda x.(x \ \ x) \ \ \lambda x.(x \ \ x))$$

Soluție. .

Secvența de reducere stânga-dreapta, a lui E, nu se termină. Conform teoremei normalizării, nicio altă secvență nu se termină. □

2. Fie o listă Scheme, precum `'(1 (2 3) () 4)`, ce poate conține alte liste drept elemente. Calculați numărul acestor liste-element, întrebuințând **funcționale** și **fără** a utiliza recursivitate explicită. Pentru exemplul de mai sus, rezultatul este 2. Soluțiile care nu respectă cele două constrângeri **nu** vor fi punctate!

Soluție. .

```
(length (filter list? '(1 (2 3) () 4)))
```

□

3. La ce se evaluează ultima expresie, în Scheme? Cum s-ar modifica rezultatul dacă `f` ar fi redefinită și aplicată în formă *curried*? Justificați!

```
(define f
  (lambda (x y)
    (+ x y)))

(f 1 (let ([x 2]) 3))
```

Soluție. .

Rezultatul este 4 în ambele situații, deoarece domeniile de vizibilitate ale celor două variabile `x` sunt izolate. □

4. Definiți obiectul `x` în Scheme, astfel încât expresia `(eq? (car (x)) (cdr (x)))` să se evalueze la `#t`.

Soluție. .

```
(define x (lambda () (cons x x)))
```

□

5. Câți parametri se vor calcula în procesul de evaluare a expresiei Haskell de mai jos? Justificați!

```
let f x y z = x + y - z in f (1 + 2) (2 + 3)
```

Soluție. .

Niciunul, deoarece evaluarea este leneșă, iar valoarea este o închidere funcțională. □

6. Sintetizați tipul funcției Haskell de mai jos. Justificați!

```
f x y = x (f y x)
```

Soluție. .

```
1 f :: a -> b -> c
2 x :: d -> e
3 a = d -> e
4 d = c
5 e = c
6 b = a
7 f :: (c -> c) -> (c -> c) -> c
```

□

7. Supraîncărcați în Haskell operatorul de verificare a egalității funcțiilor care acceptă numere drept prim parametru. Acestea vor fi considerate egale dacă valorile lor coincid în punctul 5.

Soluție. .

```
1 instance (Num a, Eq b) => Eq (a -> b) where
2     f == g = (f 5) == (g 5)
```

Barem:

- 5p: instanțiere.
- Câte 2,5p: fiecare din cele două componente ale contextului.

□

8. Definiți în Haskell fluxul șirurilor `""`, `"a"`, `"aa"`, `"aaa"`, ...

Soluție. .

```
1 strings = "" : (map ('a' :) strings)
```

□

9. Transformați în formă normală conjunctivă (FNC) următoarea propoziție:

$$\forall x.(om(x) \Rightarrow \exists y.(inima(y) \wedge are(x, y))).$$

Soluție. .

$$\forall x.(om(x) \Rightarrow \exists y.(inima(y) \wedge are(x,y)))$$
$$\forall x.(\neg om(x) \vee \exists y.(inima(y) \wedge are(x,y)))$$
$$\forall x.\exists y.(\neg om(x) \vee (inima(y) \wedge are(x,y)))$$
$$\forall x.(\neg om(x) \vee (inima(f_y(x)) \wedge are(x,f_y(x))))$$
$$\neg om(x) \vee (inima(f_y(x)) \wedge are(x,f_y(x)))$$
$$(\neg om(x) \vee inima(f_y(x))) \wedge (\neg om(x) \vee are(x,f_y(x)))$$
$$\{\neg om(x), inima(f_y(x))\}, \{\neg om(x), are(x,f_y(x))\}$$

□

10. Scrieți un program Prolog, care realizează *insertion sort*.

Soluție. .

```
insert(X, [], [X]).
insert(X, [H | T], [X , H | T]) :- X =< H, !.
insert(X, [H | T], [H | L]) :- insert(X, T, L).

isort([], []).
isort([H | T], S) :- isort(T, L), insert(H, L, S).
```

□

11. Scrieți un algoritm Markov, care dublează toate simbolurile de pe bandă. Alfabetul de bază este $A_\text{b} = \{0, 1\}$. De exemplu, pentru banda inițială `101100`, se obține `110011110000`.

Soluție. .

În programul de mai jos, `g` este o variabilă generică, iar `a`, una de lucru.

```
Double(); Ab g
    ag -> gga
    a -> .
    -> a
end
```

□

12. Scrieți un program CLIPS, care, la sfârșitul execuției, lasă faptul `(included)` în baza de cunoștințe, dacă mulțimea `A` este inclusă în mulțimea `B`. De exemplu, pentru faptele `(A 1 3 5)` și `(B 1 2 3 4 5)`, răspunsul va fi prezent, în timp ce, pentru faptele `(A 1 3 5 6)` și `(B 1 2 3 4 5)`, nu va fi.

Soluție. .

```
(deffacts sets
    (A 1 3 5 6)
    (B 1 2 3 4 5))

(defrule init
  =>
    (assert (included)))

(defrule check
    ?i <- (included)
    (A $? ?x $?)
    (not (B $? ?x $?))
  =>
    (retract ?i))
```

□