Model subiect

I. Grile

1. Pentru a instantia clasa Eq trebuie sa implementam urmatoarele functii:
a) ==
b) eq
c) ==, /=, >, >=
d) eq, not

2. Ce constrangeri de tipuri trebuie sa adaugam la functia f pentru ca urmatorul cod sa fie corect?

```
data MyData a b = MyData a b b

f :: MyData a b -> MyData a b -> Bool
f (MyData x1 y1 z1) (MyData x2 y2 z2 ) = x1 == x2 && y1 == y2 && z1 == z2
```

a) codul este deja corect;
b) Eq a
c) Eq a, Eq b
c) Eq a, Ord a

3. Se defineste:

```
data MyData a b = MyData a b b
```

Care instanta de mai jos este corecta?

```
a) instance Functor (MyData a) where
            fmap f (MyData x y z) = MyData x (f y) (f z)
b) instance Functor (MyData a) where
            fmap f (MyData x y z) = MyData (f x) (f y) (f z)
c) instance Functor (MyData a b) where
           fmap f (MyData x y z) = MyData x (f y) (f z)
d) instance Functor (MyData a b) where
           fmap f (MyData x y z) = MyData (f x) (f y) (f z)
```

4. Se defineste

```
data MyData a b = Data1 a | Data2 b
```

Care instanta de mai jos este corecta?

```
a) instance Functor (MyData a) where
           fmap f (Data1 x) = Data1 x
           fmap f (Data2 x) = Data2 (f x)
b) instance Functor (MyData a) where
           fmap f (Data1 x) = Data1 (f x)
           fmap f (Data2 x) = Data2 (f x)
c) instance Functor (MyData a b) where
           fmap f (Data1 x) = Data1 (f x)
           fmap f (Data2 x) = Data2 (f x)
d) instance Functor (MyData a b) where
           fmap f (Data1 x) = Data1 x
           fmap f (Data2 x) = f (Data2 x)
```

5. Care instanta Monoid de mai jos este *corecta*?

```
newtype MyBool = MyBool Bool
          deriving (Eq, Show)
a)   instance Semigroup MyBool where
           MyBool x <> MyBool y = MyBool ( x && y )
       instance Monoid MyBool where
            mempty = MyBool True
b) instance Monoid MyBool where
           MyBool x <> MyBool y = MyBool ( x && y )
            mempty = MyBool True
c) instance Semigroup MyBool where
           MyBool x <> MyBool y = MyBool ( x || y )
    instance Monoid MyBool where
            mempty = MyBool True
```

d) nu se poate face instanta Monoid pentru tipul MyBool.

6. Care instanta Monoid de mai jos este *corecta*?

```
newtype MyInt = MyInt Int
    deriving (Eq, Show)
a) instance Semigroup MyInt where
          MyInt x <> MyInt y = MyInt ( x + y + 1)
    instance Monoid MyInt where
          mempty = MyInt (-1)
b) instance Semigroup MyInt where
          MyInt x <> MyInt y = MyInt ( x + y + 1)
    instance Monoid MyInt where
          mempty = MyInt 0
c) instance Semigroup MyInt where
          MyInt x <> MyInt y = MyInt ( x + y + 1)
          mempty = MyInt (-1)
```

d) nu se poate face instanta Monoid pentru tipul MyInt

7. Se defineste:

```
data MyData a b = MyData a b b
```

Care instanta de mai jos este corecta?

```
a) instance Foldable (MyData a) where
    foldMap f (MyData x y z) = f y <> f z
b) instance Foldable (MyData a) where
    foldMap f (MyData x y z) = f z
c) instance Foldable (MyData a b) where
    foldMap f (MyData x y z) = f y <> f z
d) instance Foldable (MyData a) where
    foldMap f (MyData x y z) = f x  <> f y <> f z
```

8. Se defineste:

```
data MyData a b = Data1 a | Data2 b
```

Care instanta de mai jos este corecta?

```
a) instance Foldable (MyData a) where
          foldMap f (Data1 x) = mempty
```

```
        foldMap f (Data2 x) = f x
b) instance Foldable (MyData a) where
        foldMap f (Data1 x) = f x
        foldMap f (Data2 x) = f x
c) instance Foldable (MyData a) where
        foldMap f (Data1 x) = Data1 x
        foldMap f (Data2 x) = f x
d) instance Foldable (MyData a b) where
        foldMap f (Data1 x) = Data1 x
        foldMap f (Data2 x) = f x
```

9. Ce se obtine dupa instructiunea [(+ 1) , (^2)] <*> [1 , 2 , 3, 4]?
a) [2,3,4,5,1,4,9,16]
b) instructiune invalida
c) [2,3,4,5]
d) [1,4,9,16]

10. Ce se obtine dupa instructiunea (+10) <*> [1..5]?
a) instructiune invalida
b) [11, 12,13,14,15]
c) [10,20,30,40,50]
d) [1,2,3,4,5]

II. Liste

1. Sa se scrie o functie care primeste ca argumente doua siruri de caractere, si afiseaza cel mai lung prefix comun.
   f “sirulnr1” “sirdoi” = “sir”
2. Sa se scrie o functie care pentru doua liste, x si y, calculeaza suma produselor xi^2 * yi^2 cu xi din x si yi din y. Daca listele au lungimi diferite, functia va arunca o eroare.
   f [1,2,3,4] [5,6,7,8] == (1^2 * 5^2) + (2^2*6^2) + (3^2*7^2)

III. Tipuri de date

Se dau urmatoarele tipuri de date:

```
data PairInt = P Int Int deriving Show
data MyList = L  [PairInt] deriving  Show

data Exp = I Int | Add Exp Exp | Mul Exp Exp deriving Show

class MyClass m where
  toExp :: m -> Exp
```

MyList reprezinta lista de perechi, unde o pereche este reprezentata de tipul PairInt.
Exp reprezinta expresii formate din numere intregi si operatiile de adunare si inmultire.

a) Sa se scrie o instanta a clasei MyClass pentru tipul MyList astfel incat functia toExp sa transforme o lista de perechi astfel: o pereche devine adunare intre cele doua elemente, iar intre elementele listei se aplica operatia de inmultire. Pentru lista vida puteti considera ca expresia corespunzatoare este I 1.

Ex: toExp (L [P 1 2, P 2 3 , P 5 3]) == Mul (Add (I 1) (I 2)) (Mul (Add (I 2) (I 3)) (Mul (Add (I 5) (I 3)) (I 1)))

b) Sa se scrie o functie eval :: MyList -> Int care are ca parametru o lista de tipul MyList, transforma lista in expresie si apoi evalueaza expresia rezultata.