Test PP Numele ..... Grupa .....

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
a																				
b																				
c																				
d																				

1. Fie următoarele funcții Racket:

```
(define (reverse1 L R)
  (if (null? L)
      R
      (append (reverse1 (cdr L) R) (list (car L)))))
(define (reverse2 L R)
  (if (null? L)
      R
      (reverse2 (cdr L) (cons (car L) R))))
```

Alegeți afirmația corectă:

(a) `reverse1` este recursivă pe stivă, iar `reverse2` este recursivă pe coadă
(b) `reverse1` este recursivă pe coadă, iar `reverse2` este recursivă pe stivă
(c) atât `reverse1` cât și `reverse2` sunt recursive pe stivă
(d) atât `reverse1` cât și `reverse2` sunt recursive pe coadă

2. Ce este *tail call optimization*?

(a) o optimizare ce se poate aplica funcțiilor recursive pe coadă pentru a le reduce consumul de memorie datorat redimensionării stivei
(b) o optimizare ce se poate aplica funcțiilor recursive pe stivă pentru a le transforma în funcții recursive pe coadă
(c) o optimizare prin care funcțiile recursive nu folosesc deloc stiva
(d) o optimizare ce se poate aplica funcțiilor recursive pe coadă pentru a reține valorile întoarse în urmă apelurilor recursive și a le refolosi la nevoie

3. Ce va afișa următorul program Racket?

```
(define (f x) (* x 2))
(define L '(1 2 3 4))
(foldl (lambda (elem acc) (cons (f elem) acc)) '() L)
```

(a) `(8 6 4 2)`
(b) `(2 4 6 8)`
(c) `()`
(d) eroare

4. Se dă următorul program Racket:

```
(define L '(1 2 3 4 5 6 7 8))
(foldl (lambda (elem acc)
         (if (odd? elem) acc (cons elem acc)))
       '() L)
(foldr (lambda (elem acc)
         (if (even? elem) (cons elem acc) acc))
       '() L)
(filter odd? L)
(map (lambda (x) (if (even? x) x)) L)
```

Aplicația cărei funcții se va evalua la lista `'(2 4 6 8)`?

(a) `foldr`
(b) `foldl`
(c) `filter`
(d) `map`

5. La ce se va evalua următoare expresie Haskell?

```
filter id $ zipWith (<) [1, 2, 3, 4] [4, 3, 2, 1]
```

(a) `[True, True]`
(b) `[False, False]`
(c) `[True, True, False, False]`
(d) `[False, False, True, True]`

6. Ce va afișa următorul program Racket?

```
(define a 1)
(define b 1)
(define (f) (let ([b 2]) (+ a b)))
(f)
(define a 3)
(define b 3)
(f)
```

(a) `3` și `5`
(b) `2` și `6`
(c) `3` și `3`
(d) `2` și `2`

7. Fie următoarea definiție în Haskell:

```
x = 1 : (zipWith (+) x x)
```

Ce va afișa `(take 10 x)`?

(a) `[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]`
(b) `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
(c) `[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]`
(d) Programul ciclează la infinit

8. Ce va afișa următorul cod Racket?

```
(define f (delay f))
(force f)
```

(a) o promisiune
(b) o eroare
(c) o funcție
(d) Programul ciclează la infinit.

9. Ce va afișa următorul cod Racket?

```
(define x 10)
(define y (delay (+ x 10)))
(let ([x 20])
  (force y))
(define x 30)
(force y)
```

(a) `20 20`
(b) `30 40`
(c) `30 30`
(d) `20 40`

10. Ce va afișa următoarea comandă în GHCi?

```
> :t filter (==)
```

(a) Eroare de sinteză de tip
(b) Eroare de sintaxă
(c) `filter (==) :: Eq a => [a] -> [a]`
(d) `filter (==) :: [a] -> [a]`

11. Care este tipul următoarei expresii Haskell?

```
map (: []) $ map (\x -> if x then 'a' else 'b') [True]
```

(a) `[[Char]]`
(b) `[Char]`
(c) `[Bool]`
(d) Tipul nu se poate sintetiza.

12. Se dă tipul de date Haskell:

```
data MyUnion = UInt Int | UFloat Float | Complex Float Float
    deriving (Show, Eq)
```

Care dintre următoarele expresii va genera o eroare la compilare?

(a) `x = MyUnion`
(b) `x = UInt 42`
(c) `x = UFloat 42.0`
(d) `x = Complex 42.0`

13. Ce tip are următoarea funcție în Haskell?

```
f x y = head [x, y, (f x y)]
```

(a) `a -> a -> a`
(b) `a -> b -> a`
(c) `a -> b -> c`
(d) `a -> b -> [a]`

14. Se dă funcția Haskell:

```
addOne :: (Functor f, Num a) => f a -> f a
addOne = fmap (+ 1)
```

La ce se va evalua aplicația următoare?

```
(addOne . addOne) Nothing
```

(a) `Nothing`
(b) `Just Nothing`
(c) `Just 2`
(d) `Just 0`

15. Având programul următor în Haskell (prima linie poate fi ignorată):

```
{-# LANGUAGE FlexibleInstances #-}
instance (Eq a, Num a) => Eq (a -> a) where
    f == g = (f 1) == (g 1)
```

Ce va afișa apelul următor?

```
map (== (+ 1)) [(+ 1), (+ 2), (/ 2), (* 2)]
```

(a) `[True, False, False, True]`
(b) `[False, False, False, False]`
(c) `[True, False, False, False]`
(d) Eroare

16. Se dă următorul program în Prolog:

```
p(X) :- !, q(X), r(X).
p(1).
p(2).
q(3).
q(4).
r(5).
```

Cu ce se va solda încercarea de satisfacere a scopului `p(X)`?

(a) `false.`
(b) `X = 1.`
(c) `X = 2.`
(d) `X = 1; X = 2.`

17. Ce va întoarce următoarea interogare în Prolog?

```
append(X, Y, [1, 2, 3, 4]), member(1, X), length(Y, 2), !.
```

(a) `X = [1, 2], Y = [3, 4].`
(b) `false.`
(c) `true.`
(d) Eroare: `Arguments are not sufficiently instantiated`

18. Câte soluții va avea următoarea interogare în Prolog?

```
append([X | Y], Z, [1, 2, 3]).
```

(a) 3
(b) o infinitate
(c) 1
(d) 4

19. Fie următoarele definiții ale funcției f în Racket, respectiv în Haskell:

```
(define f (lambda (x) (cons x (f x))))
f = \x -> x : (f x)
```

Există vreo diferență între evaluările aplicațiilor `(car (f 1))` în Racket și `(head (f 1))` în Haskell?

(a) Da. În Racket va cicla la infinit, iar în Haskell va afișa `1`.
(b) Nu. Ambele vor cicla la infinit.
(c) Nu. Ambele vor da `1`.
(d) Nu. Ambele vor da eroare deoarece aplicăm `car`, respectiv `head`, pe o funcție.

20. Fie expresia:

```
(+ 2)
```

Există vreo diferență între rezultatele evaluării acesteia în Racket, respectiv în Haskell?

(a) Da. În Racket rezultatul este un număr, iar în Haskell, o funcție.
(b) Da. În Racket rezultatul este o funcție, iar în Haskell, un număr.
(c) Nu. În ambele limbaje rezultatul este o funcție.
(d) Nu. În ambele limbaje rezultatul este un număr.