```haskell
import Data.Maybe
import Data.List

-- Laboratorul 9

-- În acest laborator vom implementa functii pentru a lucra cu logică
-- propozitională în Haskell. Fie dată următoarea definitie:

type Nume = String
data Prop
    = Var Nume
    | F
    | T
    | Not Prop
    | Prop :|: Prop
    | Prop :&: Prop
    | Prop :->: Prop
    | Prop :<->: Prop
    deriving Eq
infixr 2 :|:
infixr 3 :&:

-- Tipul Prop este o reprezentare a formulelor propozitionale.
-- Variabilele propozitionale, precum p si q pot fi reprezentate ca
-- Var "p" si Var "q". În plus, constantele booleene F si T reprezintă
-- false si true, operatorul unar Not reprezintă negatia (¬; a nu se
-- confunda cu functia not :: Bool -> Bool) si operatorii (infix)
-- binari :|: si :&: reprezintă disjunctia (∨) si conjunctia (∧).

-- Exercitiul 1

-- Scrieti următoarele formule ca expresii de tip Prop, denumindu-le p1, p2, p3.

-- 1. (P ∨ Q) ∧ (P ∧ Q)

p1 :: Prop
p1 = (Var "P" :|: Var "Q") :&: (Var "P" :&: Var "Q")

-- 2. (P ∨ Q) ∧ (¬P ∧ ¬Q)

p2 :: Prop
p2 = (Var "P" :|: Var "Q") :&: (Not (Var "P") :&: Not (Var "Q"))

-- 3. (P ∧ (Q ∨ R)) ∧ ((¬P ∨ ¬Q) ∧ (¬P ∨ ¬R))
```

```haskell
p3 :: Prop
p3 = (Var "P" :&: (Var "Q" :|: Var "R")) :&: ((Not (Var "P") :|: Not (Var "Q"))
:&: (Not (Var "P") :|: Not (Var "R")))

-- Exercitiul 2

-- Faceti tipul Prop instantă a clasei de tipuri Show, înlocuind conectivele
-- Not, :|: si :&: cu ~, | si & si folosind direct numele variabilelor în loc de
-- constructia Var nume.

instance Show Prop where
    show :: Prop -> String
    show (Var p) = p
    show F = "false"
    show T = "true"
    show (Not p) = "(~" ++ show p ++ ")"
    show (p :|: q) = "(" ++ show p ++ " | " ++ show q ++ ")"
    show (p :&: q) = "(" ++ show p ++ " & " ++ show q ++ ")"
    show (p :->: q) = "(" ++ show p ++ " -> " ++ show q ++ ")"
    show (p :<->: q) = "(" ++ show p ++ " <-> " ++ show q ++ ")"

test_ShowProp :: Bool
test_ShowProp = show (Not (Var "P") :&: Var "Q") == "((~P) & Q)"

-- Evaluarea expresiilor logice

-- Pentru a putea evalua o expresie logică vom considera un mediu de evaluare
-- care asociază valori Bool variabilelor propozitionale:

type Env = [(Nume, Bool)]

-- Tipul Env este o listă de atribuiri de valori de adevăr pentru (numele)
-- variabilelor propozitionale. Pentru a obtine valoarea asociată unui Nume în
-- Env, putem folosi functia predefinită:

-- lookup :: Eq a => a -> [(a,b)] -> Maybe b

-- Desi nu foarte elegant, pentru a simplifica exercitiile de mai jos, vom
-- definit o variantă a functiei lookup care generează o eroare dacă valoarea nu
-- este găsită.

impureLookup :: Eq a => a -> [(a,b)] -> b
impureLookup a = fromJust . lookup a

-- O solutie mai elegantă ar fi să reprezentăm toate functiile ca fiind partiale
```

```haskell
-- (rezultat de tip Maybe) si sa controlam propagarea erorilor.

-- Exercitiul 3

-- Definiti o functie eval care dat fiind o expresie logică si un mediu de
-- evaluare, calculează valoarea de adevăr a expresiei.

env :: Env
env = [("P", False), ("P", True)]

eval :: Prop -> Env -> Bool
eval T _ = True
eval F _ = False
eval (Var p) env = impureLookup p env
eval (Not p) env = not (eval p env)
eval (p :&: q) env =  eval p env && eval q env
eval (p :|: q) env =  eval p env || eval q env
eval (p :->: q) env = not (eval p env) || eval q env
eval (p :<->: q) env = eval (p :->: q) env && eval (q :->: p) env

test_eval :: Bool
test_eval = eval (Var "P" :|: Var "Q") [("P", True), ("Q", False)] == True

-- Satisfiabilitate

-- O formulă în logica propozitională este satisfiabilă dacă există o atribuire
-- de valori de adevăr pentru variabilele propozitionale din formulă pentru care
-- aceasta se evaluează la True.

-- Pentru a verifica dacă o formulă este satisfiabilă vom genera toate
-- atribuirile posibile de valori de adevăr si vom testa dacă formula se
-- evaluează la True pentru vreuna dintre ele.

-- Exercitiul 4

-- Definiti o functie variabile care colectează lista tuturor variabilelor dintr-
-- o formulă.
-- Indicatie: folositi functia nub.
-- nub (meaning "essence") removes duplicates elements from a list

variabile :: Prop -> [Nume]
variabile F = []
variabile T = []
variabile (Var p) = [p]
variabile (Not p) = variabile p
```

```haskell
variabile (p :&: q) = nub (variabile p ++ variabile q)
variabile (p :|: q) = nub (variabile p ++ variabile q)
variabile (p :->: q) = nub (variabile p ++ variabile q)
variabile (p :<->: q) = nub (variabile p ++ variabile q)

test_variabile :: Bool
test_variabile = variabile (Not (Var "P") :&: Var "Q") == ["P", "Q"]

-- Exercitiul 5

-- Dată fiind o listă de nume, definiti toate atribuirile de valori de
-- adevăr posibile pentru ea.

-- Fac o functie auxiliara pentru a genera produs cartezian de valori Bool
-- ex: valori_adevar 2 = [[True, True], [False, False], [False, True]]

valori_adevar :: Int -> [[Bool]]
valori_adevar 0 = []
valori_adevar 1 = [[False], [True]]
valori_adevar n = [False : xs | xs <- valori_adevar(n-1)] ++ [True : xs | xs <-
valori_adevar(n-1)]

envs :: [Nume] -> [Env]
envs variabile = map (zip variabile) (valori_adevar(length variabile))

test_envs :: Bool
test_envs = envs ["P", "Q"] == [[("P", False), ("Q", False)], [("P", False),
("Q", True)], [("P", True), ("Q", False)], [("P", True), ("Q", True)]]

-- Exercitiul 6

-- Definiti o functie satisfiabila care dată fiind o Propozitie verifică dacă
-- aceasta este satisfiabilă. Puteti folosi rezultatele de la exercitiile 4 si 5.

satisfiabila :: Prop -> Bool
satisfiabila prop = any (eval prop) (envs (variabile prop))

test_satisfiabila1 :: Bool
test_satisfiabila1 = satisfiabila (Not (Var "P") :&: Var "Q") == True

test_satisfiabila2 :: Bool
test_satisfiabila2 = satisfiabila (Not (Var "P") :&: Var "P") == False
```

```haskell
-- Exercitiul 7

-- O propozitie este validă dacă se evaluează la True pentru orice interpretare a
-- varibilelor.O forumare echivalenta este aceea că o propozitie este validă dacă
-- negatia ei este nesatisfiabilă. Definiti o functie valida care verifică dacă o
-- propozitie este validă.

valida :: Prop -> Bool
valida prop = all (eval prop) (envs (variabile prop))

test_valida1 :: Bool
test_valida1 = valida (Not (Var "P") :&: Var "Q") == False

test_valida2 :: Bool
test_valida2 = valida (Not (Var "P") :|: Var "P") == True

-- Implicatie si echivalentă

-- Exercitiul 9

-- Extindeti tipul de date Prop si functiile definite până acum pentru a include
-- conectivele logice -> (implicatia) si <-> (echivalenta), folosind
-- constructorii :->: si :<->:.

-- Am modificat deasupra.

-- Exercitiul 10

-- Două propozitii sunt echivalente dacă au mereu aceeasi valoare de adevăr,
-- indiferent de valorile variabilelor propozitionale. Scrieti o functie care
-- verifică dacă două propozitii sunt echivalente.

echivalenta :: Prop -> Prop -> Bool
echivalenta prop1 prop2 = valida (prop1 :<->: prop2)

test_echivalenta1 :: Bool
test_echivalenta1 = True == (Var "P" :&: Var "Q") `echivalenta` (Not (Not (Var
"P") :|: Not (Var "Q")))

test_echivalenta2 :: Bool
test_echivalenta2 = False == (Var "P") `echivalenta` (Var "Q")

test_echivalenta3 :: Bool
test_echivalenta3 = True == (Var "R" :|: Not (Var "R")) `echivalenta` (Var "Q"
:|: Not (Var "Q"))
```