Programare Declarativa — Test 2

Nume: _____ grupa:_____ 17 jan 2017

Se dă tipul de date:

```
data Expr = Var String
          | Const Int
          | Neg Expr
          | Expr Plus Expr
          | Expr Times Expr
```

1. Să se scrie o funcție `transform` care ia ca argument o funcție `t` de transformare (locală) și o expresie e de tip `Expr` și întoarce expresia obținută prin aplicarea funcției `t` fiecărei subexpresii a lui e, de jos în sus.

```
transform :: (Expr -> Expr) -> Expr -> Expr
transform (\x -> Neg x) (Plus (Neg (Const 4)) (Var "x")) ==
  Neg (Plus (Neg (Neg (Neg (Const 4)))) (Neg (Var "x")))
transform (\x -> Neg x) (Times (Const 7) (Const 8)) ==
  Neg (Times (Neg (Const 7)) (Neg (Const 8)))
```

2. Să se scrie o funcție de transformare locală `subLocal` care ia ca argument o substituție (funcție care asociază expresii pentru nume de variabile), și o expresie, și dacă expresia e o variabilă o transformă conform substituției, altfel o lasă neschimbată.

```
subLocal :: (String -> Expr) -> Expr -> Expr
x1y2 name = if name == "x" then Const 1 else Const 2
subLocal x1y2 (Var "x") == Const 1
subLocal x1y2 (Const 7) == Const 7
```

3. Folosiți transform și `subLocal` pentru a implementa o funcție `substitute` care ia ca argument o substituție și o expresie și înlocuiește toate variabilele care apar în expresie conform substituției.

```
substitute :: (String -> Expr) -> Expr -> Expr
substitute x1y2 (Var "x") == Const 1
substitute x1y2(Neg (Var "x")) == Neg (Const 1)
substitute x1y2 (Plus (Var "x") (Var "y")) == Plus (Const 1) (Const 2)
substitute x1y2 (Neg (Times (Plus (Const 7) (Var "x")) (Var "y"))) ==
  Neg (Times (Plus (Const 7) (Const 1)) (Const 2))
```

4. Să se scrie o funcție de transformare locală reduce care simplifică operația din vârful expresiei date ca argument presupunând că subexpresiile sunt deja simplificate.

```
reduce :: Expr -> Expr
reduce (Plus (Const 0) (Const 7))  == Const 7
reduce (Times (Const 7) (Const 1)) == Const 7
reduce (Neg (Neg (Const 7)))       == Const 7
```

5. Folositi funcțiile de mai sus pentru a defini o functie `simplify` care ia ca argument o substituție si o expresie oarecare și o simplifică cât de mult se poate.

```
simplify :: (String -> Expr) -> Expr -> Expr
simplify x1y2 (Neg (Neg (Var "x"))) == Const 1
simplify x1y2 (Neg (Plus (Var "x") (Const 0))) == Neg (Const 1)
simplify x1y2 (Times (Const 1) (Plus (Var "y") (Const 0))) == Const 2
```