Se dă tipul de date:
```
data Exp = Var String        -- Variabilă   x
         | Val Int           -- Întreg      5
         | Op Exp String Exp -- Operaţie 3 <= x
         | If Exp Exp Exp    -- Conditional if (3 <= x) then 3 else x
         | Lambda String Exp -- Funcţie anonimă \ x -> 3 + x
```

Cerinţe:

1. Să se scrie o funcţie `subExps` care ia ca argument o expresie şi întoarce lista subexpresiilor ei directe.
   ```
   subExps :: Exp -> [Exp]
   ```

2. Să se scrie o funcţie `aggExps` care ia ca argument o expresie `e` şi o listă de expresii `es` şi înlocuieşte subexpresiile directe ale lui `e` cu cele din `es`, în ordine, dacă nr. subexpresiilor coincide cu lungimea lui `es`; dacă nu, expresia rămâne neschimbată. (`aggExps` e un fel de inversă a lui `subExps`)
   ```
   aggExps :: Exp -> [Exp] -> Exp
   ```

3. Să se scrie o funcţie `foldExp` care ia ca argument o funcţie (de agregare) `f` şi o expresie `e` şi calculează o valoare astfel: calculează recursiv valorile corespunzătoare subexpresiilor directe ale lui `e`, apoi cheamă funcţia `f` cu expresia `e` şi lista de valori obţinută pentru a obţine valoarea finală. Sugestie: folosiţi funcţia `subExps`.
   ```
   foldExp :: (Exp -> [b] -> b) -> Exp -> b
   ```

4. Să se scrie o funcţie `transform` care ia ca argument o funcţie `t` de transformare (locală) şi o expresie `e` şi întoarce expresia obţinută prin aplicarea funcţiei `t` fiecărei subexpresii a lui `e`, de jos în sus. Sugestie: folosiţi `foldExp` şi `aggExps`
   ```
   transform :: (Exp -> Exp) -> Exp -> Exp
   ```

5. Să se scrie o funcţie `freeVars` care ia ca argument o expresie `e` şi calculează lista variabilelor libere care apar în `e`. Pentru toţi constructorii se reunesc variabilele din subexpresii, cu excepţia lui `Lambda` pentru care se elimină variabila definită. Sugestie: folosiţi `foldExp` şi o funcţie de agregare nouă.

```
freeVars :: Exp -> [String]
```