Laboratorul 2

Limbajul unui limbaj imperativ simplu

Definim în Haskell limbajul IMP:

```
type Name = String

data  Pgm  = Pgm [Name] Stmt
        deriving (Read, Show)

data Stmt = Skip | Stmt ::: Stmt | If BExp Stmt Stmt | While BExp Stmt | Name := AExp
        deriving (Read, Show)

data AExp = Lit Integer | AExp :+: AExp | AExp :*: AExp | Var Name
        deriving (Read, Show)

data BExp = BTrue | BFalse | AExp :==: AExp | Not BExp
        deriving (Read, Show)

infixr 2 :::
infix 3 :=
infix 4 :==:
infixl 6 :+:
infixl 7 :*:

type Env = [(Name, Integer)]
```

Dorim ca un program să afișeze starea memoriei la sfarsitul executiei programului. Toate variabilele trebuie să fie inițializate înainte de a fi folosite. Variabilele din lista inițială sunt inițializate cu `0`.

De exemplu, programul

```
factStmt :: Stmt
factStmt =
 "p" := Lit 1 ::: "n" := Lit 3:::
 While (Not (Var "n" :==: Lit 0))
   ( "p" := Var "p" :*: Var "n" :::
     "n" := Var "n" :+: Lit (-1)
   )

pg1 = Pgm [] factStmt
```

va afișa lista `[("n",0),("p",6)]`

Exercițiu

Să se definească funcțiile de evaluare a expresiilor de tip `Pgm`, `Stmt`, `AExp` și `BExp`.

```
pEval :: Pgm -> Env
sEval :: Stmt -> Env -> Env
bEval :: BExp -> Env -> Bool
aEval :: AExp -> Env -> Integer
```