Eval1.hs

```haskell
module Eval1 (eval) where

import AST

-- Estados
type State = [(Variable,Int)]

-- Estado nulo
initState :: State
initState = []

-- Busca el valor de una variable en un estado, asumo que aparece
exactamente una vez en el estado
lookfor :: Variable -> State -> Int
lookfor x [(_,i)] = i
lookfor x ((y,i):xs) = if x==y then i else lookfor x xs


-- Cambia el valor de una variable en un estado
update :: Variable -> Int -> State -> State
update x i [] = [(x,i)]
update x i ((y,j):xs) = if x==y then ((x,i):xs) else (y,j):(update x i
xs)

-- Evalua un programa en el estado nulo
eval :: Comm -> State
eval p = evalComm p initState

-- Evalua un comando en un estado dado
evalComm :: Comm -> State -> State
evalComm Skip s = s
evalComm (Let x ie) s = update x (evalIntExp ie s) s
evalComm (Seq c1 c2) s = evalComm c2 (evalComm c1 s)
evalComm (Cond be c1 c2) s = case (evalBoolExp be s) of
                                True -> evalComm c1 s
                                False -> evalComm c2 s
evalComm w@(While be c) s = case (evalBoolExp be s) of
                                True -> evalComm (Seq c w) s
                                False -> s

-- Evalua una expresion entera, sin efectos laterales
evalIntExp :: IntExp -> State -> Int
evalIntExp (Const i) s = i
evalIntExp (Var x) s = lookfor x s
evalIntExp (UMinus ie) s = (-1) * (evalIntExp ie s)
evalIntExp (Plus ie1 ie2) s = (evalIntExp ie1 s) + (evalIntExp ie2 s)
evalIntExp (Minus ie1 ie2) s = (evalIntExp ie1 s) - (evalIntExp ie2 s)
evalIntExp (Times ie1 ie2) s = (evalIntExp ie1 s) * (evalIntExp ie2 s)
evalIntExp (Div ie1 ie2) s = div (evalIntExp ie1 s) (evalIntExp ie2 s)
```

```haskell
-- Evalua una expresion entera, sin efectos laterales
evalBoolExp :: BoolExp -> State -> Bool
evalBoolExp BTrue s = True
evalBoolExp BFalse s = False
evalBoolExp (Eq ie1 ie2) s = (evalIntExp ie1 s) == (evalIntExp ie2 s)
evalBoolExp (Lt ie1 ie2) s = (evalIntExp ie1 s) < (evalIntExp ie2 s)
evalBoolExp (Gt ie1 ie2) s = (evalIntExp ie1 s) > (evalIntExp ie2 s)
evalBoolExp (And be1 be2) s = (evalBoolExp be1 s) && (evalBoolExp be2 s)
evalBoolExp (Or be1 be2) s = (evalBoolExp be1 s) || (evalBoolExp be2 s)
evalBoolExp (Not be) s = not (evalBoolExp be s)
```