```haskell
-- setting the "warn-incomplete-patterns" flag asks GHC to warn you
-- about possible missing cases in pattern-matching definitions
{-# OPTIONS_GHC -fwarn-incomplete-patterns #-}

-- see https://wiki.haskell.org/Safe_Haskell
{-# LANGUAGE Safe #-}

module Interpreter (run, Storage, emptyStorage, update) where

import AbstractSyntax
import IOPrime

type Storage = Identifier -> Integer

emptyStorage :: Storage
emptyStorage i = error ("Uninitialized identifier " ++ i)

update :: Identifier -> Integer -> Storage -> Storage
update i x m = m'
 where
   m' :: Storage
   m' j | i == j    = x
        | otherwise = m j

number :: Bool -> Integer
number False = 0
number True  = 1

boolean :: Integer -> Bool
boolean 0 = False
boolean _ = True

opEval :: OpName -> [Integer] -> Integer
opEval Add     [x, y] = x + y
opEval Sub     [x, y] = x - y
opEval Mul     [x, y] = x * y
opEval Div     [x, y] = x `div` y
opEval Mod     [x, y] = x `mod` y
opEval Eq      [x, y] = number(x == y)
opEval Leq     [x, y] = number(x <= y)
opEval Less    [x, y] = number(x <  y)
opEval Geq     [x, y] = number(x >= y)
opEval Greater [x, y] = number(x >  y)
opEval And     [x, y] = number(boolean x && boolean y)
opEval Or      [x, y] = number(boolean x || boolean y)
opEval Not     [x]    = number(not(boolean x))
opEval op      xs     = error ("Interpreter bug. "
                            ++ "Please contact the software maintainer. "
                            ++ "Tried to apply " ++ show op
                            ++ " to " ++ show xs)

eval :: Storage -> Expr ->  Integer
eval m (Constant x) = x
eval m (Var i)      = m i
eval m (Op o es)    = opEval o [eval m e | e <- es]

--------------------------------------------------------------------------------
---------------- DO **NOT** MAKE ANY CHANGES ABOVE THIS LINE --------------------
--------------------------------------------------------------------------------
```

```
run :: Program -> Storage -> IO' Storage

run (i := e)          m = return (update i (eval m e) m)

run (IfElse e p q)    m | boolean (eval m e) = run p m
                        | otherwise          = run q m

run (If e p)          m | boolean (eval m e) = run p m
                        | otherwise          = return m

run (While e p)       m | boolean (eval m e) = do
                                               m' <- run p m
                                               run (While e p) m'
                        | otherwise          = return m

run (Block [])        m = return m
run (Block (p : ps))  m = do
                            m' <- run p m
                            run (Block ps) m'

run (Read i)          m = do
                            value <- getLine'
                            return (update i (read value) m)


--     main = do
        -- putStrLn "Hello, what's your name?"
        -- name <- getLine
        -- putStrLn ("Hey " ++ name ++ ", you rock!")
run (Write e)         m = do
                            putStrLn' (show (eval m e))
                            return m
run (Print s)         m = do
                            putStrLn' s
                            return m

run (For i mn mx p)   m = do
                            let minn = eval m mn
                            let m' = update i minn m
                            run2 (For i mn mx p) m'


             where
                 run2 :: Program -> Storage -> IO' Storage
                 run2 (For i mn mx p) m = if boolean (eval m (Op Leq [(Var i),
mx])) then do
                                                                              m' <-
run p m
                                                                              let
newi = 1 + (eval m' (Var i))
                                                                              let m''
= (update i newi m')
                                                                              (run2
(For i mn mx p) m'')
                                          else return m
                 run2 _ m = return m
```