```
-- 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 Optimization (optimize) where

import Control.Monad.State

import AbstractSyntax
import Interpreter

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

type OptStorage = Identifier -> Maybe Integer

emptyOptStorage :: OptStorage
emptyOptStorage i = Nothing

deleteVar :: Identifier -> OptStorage -> OptStorage
deleteVar i m j | i == j    = Nothing
                | otherwise = m j

deleteVars :: [Identifier] -> OptStorage -> OptStorage
deleteVars [] m = m
deleteVars (i:is) m = deleteVars is (deleteVar i m)


-- A pair of stateful monadic computations optimizing expression and
-- programs in the presense of an OptStorage associating *constant*
-- variables to their values ....

optExpr :: Expr -> State OptStorage Expr
optExpr = undefined

optProgram  :: Program -> State OptStorage Program
optProgram = undefined

-- Identifier := Expr
--               | Block [Program]
--               | While Expr Program
--               | If Expr Program
--               | IfElse Expr Program Program
--               | Read Identifier
--               | Write Expr
--               | Print String
--               | For Identifier Expr Expr Program

-- Replace this with any implementation you like
optimize :: Program -> Program
optimize = undefined

-- The suggested implementation for using the above monadic setup:
-- optimize p = fst $ runState (optProgram p) emptyOptStorage
```