# Interpreter Activity 1

Mattox Beckman

## Code

Here is part of the code for the `i3.hs` interpreter.

```haskell
-- The Types

data Val = IntVal Integer
   deriving (Show,Eq)

data Exp = IntExp Integer
         | IntOpExp String Exp Exp
   deriving (Show,Eq)

type Env = [(String,Exp)]

-- Evaluator

intOps = [ ("+",(+))
         , ("-",(-))
         , ("*",(*))
         , ("/",div)]

liftIntOp f (IntVal i1) (IntVal i2) = IntVal (f i1 i2)
liftIntOp f _           _           = 0

eval :: Exp -> Env -> Val
eval (IntExp i) _ = IntVal i

eval (IntOpExp op e1 e2) env =
  let v1 = eval e1 env
      v2 = eval e2 env
      Just f = lookup op intOps
  in liftIntOp f e1 e2
```

1. With a partner, code review this. Two lines have errors! Find them and correct them.

2. Add variables to this. To do this you need to add a constructor to `Exp` and a clause to `eval`.

3. If there's time: add comparison operations to the language. You will need a separate variable `compOps` to do this, another constructor for `Exp`, and another clause for `eval`. You may need another lifting function as well. Why can't you just combine this with `intOps`?