CS 440 — Interpreter Activity

Code

```haskell
-- ----------------------------------------
-- Type Declarations
-- ----------------------------------------

data Stmt = ExpStmt Exp
          | SetStmt String Exp
     deriving (Show,Read)

data Exp = IntExp Int
         | VarExp String
         | OpExp String Exp Exp
         | LamExp String Exp
         | AppExp Exp Exp
     deriving (Show,Read)

data Val = IntVal Int
         | CloVal String Exp Env
     deriving (Show,Read)

type Env = [ (String, Val) ]

data State = State String Env

-- ----------------------------------------
-- Operators
-- ----------------------------------------

lift op v1 v2 =
  let IntVal i1 = v1
      IntVal i2 = v2
  in IntVal (op i1 i2)

intOpList = [ ("+", (+)) ]
```

```haskell
-- ----------------------------------------
-- Eval
-- ----------------------------------------

eval (IntExp i) env = IntVal i
eval (VarExp s) env =
  case lookup s env of
    Just v -> v
    Nothing -> IntVal 0
eval (LamExp s body) env = CloVal s body env
eval (OpExp o e1 e2) env =
  let v1 = eval e1 env
      v2 = eval e2 env
      op = lookup o intOpList
  in case op of
    Just op' -> lift op' v1 v2
    Nothing -> error "Operator not found."
eval (AppExp e1 e2) env =
  let v1 = eval e1 env
      v2 = eval e2 env
  in case v1 of
    CloVal s b env' -> eval b ((s,v2) : env')
    _               -> error "Not a function."

-- ----------------------------------------
-- Exec
-- ----------------------------------------

exec (SetStmt s exp) env =
    State (s ++ " set.") ((s,eval exp env):env)
exec (ExpStmt exp) env =
    State (show result) env
  where result = eval exp env

-- ----------------------------------------
-- REPL
-- ----------------------------------------

repl env = do
  putStr "> "
  str <- getLine
  let State result nuenv = exec (read str) env
  putStrLn result
  putStrLn ""
  repl nuenv

main = repl [ ("x",IntVal 1) ]
```

Sample Run

```
> ExpStmt (VarExp "x")
IntVal 1

> SetStmt "y" (OpExp "+" (IntExp 1) (VarExp "x"))
y set.

> ExpStmt (VarExp "y")
IntVal 2

> ExpStmt (AppExp (LamExp "x" (OpExp "+" (VarExp "x") (IntExp 40))) (IntExp 2))
IntVal 42
```

Questions

1. In groups of two or three, each person take turns explaining one of the code sections above.
2. Explain the steps the code takes to get the answer 42 in the example above.
3. Show how to add subtraction to this interpreter.
4. Show how to add let statements to this interperter. Use the form $LetExp\ var\ value\ body$.
5. Bonus: Show how to allow functions of more than one argument.