Here is part of the code for the i4.hs interpreter.

```haskell
data Val = IntVal Integer
  deriving (Show, Eq)

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

type Env = [(String, Val)]

intOps = [('+','+'), ('-','-'), ('*','*'), ('/','/')]

liftIntOp f (IntVal i1) (IntVal i2) = IntVal (f i1 i2)
liftIntOp f _ _ = IntVal 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 v1 v2

eval (VarExp v) env = 
  case lookup v env of 
    Just vv -> vv
    Nothing -> IntVal 0

eval (LetExp var e1 e2) env = 
  let v1 = eval e1 env 
    in eval e2 (var,v1):env
```

<table>
<thead>
<tr>
<th>Manager</th>
<th>Keeps team on track</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recorder</td>
<td>Records decisions</td>
</tr>
<tr>
<td>Reporter</td>
<td>Reports to class</td>
</tr>
<tr>
<td>Reflector</td>
<td>Assesses team performance</td>
</tr>
</tbody>
</table>
**Problem 1)** Code review this. Two lines have errors, and they are different ones than from last time! Find them and correct them.

**Problem 2)** Consider the following code:

```haskell
1 Prelude> delta = 1
2 Prelude> inc x = x + delta
3 Prelude> inc 10
4 11
5 Prelude> messup x = let delta = 2 in inc x
6 Prelude> messup 10
7   -- What happens?
```

Have each member of the team predict the output of `messup 10`. Come to a consensus about the output and why.

### Adding functions to the Interpreter

Consider the following code that would add functions to the language. For the data types we are only showing the added clauses to save space.

```haskell
1 data Val = FunVal String Exp
2
3 data Exp = FunExp String Exp
4       | FunApp Exp Exp
5
6 eval (FunExp var body) env = FunVal var body
7 eval (FunApp e1 e2) env =
8   let FunVal var body = eval e1 env
9       arg = eval e2 env
10     in eval body ((var, arg):env)
```

**Problem 3)** The constructor for `FunVal` takes an `Exp` for the body, not a `Val`. Why do we not evaluate the body of the function when we create the `FunVal`?

**Problem 4)** What does the syntax `((var, arg):env)` indicate?
**Problem 5)** Here is a reasonably equivalent program to the Haskell one above.

```haskell
Prelude> eval (LetExp "delta" (IntExp 1))
    (LetExp "inc" (FunVal "x" (IntOpExp "+" (VarExp "x") (VarExp "delta")))
    (LetExp "messup" (FunVal "x" (LetExp "delta" (IntExp 2) (FunApp (VarExp "inc") (VarExp "x"))))
    (FunApp (VarExp "messup") (IntExp 10))))
```

Does it give the same result as the Haskell program? Hint: the answer is `no." What goes wrong?

### Closures

**Problem 6** The instructor will talk briefly about *Closures*, which fix this problem. Modify the given code to implement closures.
### Interpreter Activity 2--- Reflector's Report

<table>
<thead>
<tr>
<th>Role</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manager</td>
<td>Keeps team on track</td>
</tr>
<tr>
<td>Recorder</td>
<td>Records decisions</td>
</tr>
<tr>
<td>Reporter</td>
<td>Reports to Class</td>
</tr>
<tr>
<td>Reflector</td>
<td>Assesses team performance</td>
</tr>
</tbody>
</table>

1. What was a strength of your team's performance for this activity?

2. What could you do next time to increase your team's performance?

3. What insights did you have about the activity or your team's interaction today?