Assignment 7 solutions

```haskell
{-# LANGUAGE TypeSynonymInstances #-}
{-# LANGUAGE FlexibleInstances #-}
{-# LANGUAGE FlexibleContexts #-}

import Control.Monad.Writer
import Control.Monad.State
import Control.Monad.Identity

class Monad m => ArithMonad m where
  add  :: Integer -> Integer -> m Integer
  mult :: Integer -> Integer -> m Integer
  divi :: Integer -> Integer -> m Integer

twiceAndOne x = do
  twiceX <- mult 2 x
  add twiceX 1
twiceAndOneA x = mult 2 x >>= \y -> add 1 y
twiceAndOneB x = mult 2 x >>= add 1

grak :: ArithMonad m => Integer -> Integer -> m Integer
grak x y = do
  a <- mult x x  -- x^2
  b <- mult 3 a  -- 3x^2
  c <- mult x y  -- xy
  d <- mult 2 c  -- 2xy
  e <- mult y y  -- y^2
  f <- mult 4 e  -- 4y^2
  g <- add b d   -- 3x^2 + 2xy
  h <- add g f   -- 3x^2 + 2xy + 4y^2
  i <- add h (-5) -- 3x^2 + 2xy + 4y^2 - 5
  return i

instance ArithMonad Identity where
  add x y = return (x+y)
  mult x y = return (x*y)
  divi x y = return (div x y)

instance ArithMonad (Writer String) where
  add x y = do
    tell $ show x ++ " + " ++ show y ++ "\n"
```

```
    return (x+y)
  mult x y = do
    tell $ show x ++ " * " ++ show y ++ "\n"
    return (x*y)
  divi x y = do
    tell $ show x ++ " / " ++ show y ++ "\n"
    return (div x y)

runLog action = do
  let (result, log) = runWriter action
  putStr log
  return result

slowExp _ 0 = 1
slowExp x y = x * slowExp x (y-1)

slowExpM _ 0 = return 1
slowExpM x y = do
  yMinus1 <- add y (-1)
  recurse <- slowExpM x yMinus1
  mult x recurse

fastExp _ 0 = 1
fastExp x y
  | even y = fastExp (x*x) (div y 2)
  | otherwise = x * fastExp x (y-1)

fastExpM _ 0 = return 1
fastExpM b e
  | even e = do
      bsq <- mult b b
      e2 <- divi e 2
      fastExpM bsq e2
  | otherwise = do
      e1 <- add e (-1)
      r <- fastExpM b e1
      mult b r

instance ArithMonad (State Int) where
  add x y = modify succ >> return (x+y)
  mult x y = modify succ >> return (x*y)
  divi x y = modify succ >> return (div x y)

runCount :: State Int a -> (a, Int)
runCount action = runState action 0
```

```
main = flip execStateT (0,0) $ do
    verify "1.01"  27 $ runIdentity $ twiceAndOne 13
    verify "1.02" 165 $ runIdentity $ twiceAndOne 82
    verify "2.01" 326 $ runIdentity $ grak 3 8
    verify "2.02" 215 $ runIdentity $ grak 6 4
    verify "3.01" (165, "2 * 82\n164 + 1\n")
      $ runWriter $ twiceAndOne 82
    -- Actually, the order of your logged operations could slightly
    -- differ for this one, and still be correct:
    verify "3.02" (215, "6 * 6\n3 * 36\n6 * 4\n2 * 24\n4 * 4\n4 * 16\n108 + 48\n156 +
      $ runWriter $ grak 6 4
    verify "4.01" 32 $ runIdentity $ slowExpM 2 5
    verify "4.02" 32 $ runIdentity $ fastExpM 2 5
    let answer = 1234766956912479358262297818562 56
    verify "4.03" answer $ runIdentity $ slowExpM 14 28
    verify "4.04" answer $ runIdentity $ fastExpM 14 28
    verify "5.01" (32,10) $ runCount $ slowExpM 2 5
    verify "5.02" (32, 8) $ runCount $ fastExpM 2 5
    verify "5.03" (answer,56) $ runCount $ slowExpM 14 28
    verify "5.04" (answer,14) $ runCount $ fastExpM 14 28
  where
    say = liftIO . putStrLn
    correct (k, n) = (k+1, n+1)
    incorrect (k, n) = (k, n+1)
    assert s = verify s True
    verify :: (Show a, Eq a) => String -> a -> a -> StateT (Int,Int) IO ()
    verify = verify' (==)
    verifyF = verify' (\x y -> abs(x-y) < 0.00001)
    verify' :: (Show a) => (a -> a -> Bool) -> String -> a -> a ->
               StateT (Int,Int) IO ()
    verify' eq tag expected actual
      | eq expected actual = do
          modify correct
          say $ " OK " ++ tag
      | otherwise = do
          modify incorrect
          say $ "ERR " ++ tag ++ ": expected " ++ show expected
            ++ " got " ++ show actual
-- End of test driver
```