Assignment 4

due *Tue 1 Oct*

```haskell
{-# LANGUAGE FlexibleContexts #-}
module A04 where

import Control.Monad.Writer
import Data.Char
import GHC.Stack
import System.Exit

{- Here is an enumeration type for the 12 months of the year. It
 derives Eq, Ord (so you can compare equality and ordering),
 Show (for converting to a string), Enum (for conversion
 to/from integers and ranges), and Bounded (defining a minimum
 and maximum value). Try these:

     ghci> fromEnum May
     4

 Is that surprising? With derived Enum, first element is at
 zero.

     ghci> toEnum 11 :: Month
     Dec
     ghci> minBound :: Month
     Jan
     ghci> maxBound :: Month
     Dec
-}

data Month
  = Jan | Feb | Mar
  | Apr | May | Jun
  | Jul | Aug | Sep
  | Oct | Nov | Dec
  deriving (Eq, Show, Enum, Ord, Bounded)

{- Here is a type and constructor that pairs together a month
 and a day number to represent a calendar day. The Eq and Ord
 instances do sensible things. The derived Show instance uses
 the same Haskell syntax you'd use to create the day value, it
 just places it into a string:

     ghci> show (CalDay Mar 9)
     "CalDay Mar 9"

 Haskell can derive a Bounded instance because both parts of
 the pair (Month and Int) are also Bounded. But unfortunately,
 the derived instance doesn't do what we'd want:

     ghci> minBound :: CalDay
     CalDay Jan (-9223372036854775808)

 It is pairing the minimal month with the minimal Int! We'll
 fix that in a moment...

-}

data CalDay =
  CalDay Month Int
  deriving (Eq, Ord, Show, Bounded)

{- First let's improve the readability of the Show result.
 Uncomment the two-line manual instance declaration below, AND
 REMOVE Show from the list of derived instances ABOVE.
 (Otherwise the compiler will report duplicate instances for
 the CalDay type.)

 Replace the "??" definition below so that days are formatted
 like "Sep-27", or if the day is one digit it should have a
 leading zero, like "Oct-01".
-}

-- instance Show CalDay where
--   show (CalDay month day) = "??"

{- Now let's improve the correctness of the Bounded instance.
 Uncomment the three-line instance declaration below, AND
 REMOVE Bounded from the list of derived CalDay instances
 ABOVE.
 I put some arbitrary dates in there to make it compile, but
 obviously the minimum should correspond to January 1st and
 the maximum to December 31st.
-}

-- instance Bounded CalDay where
--   minBound = CalDay Jan 13
--   maxBound = CalDay Feb 14

{- This function should report the number of days in a given
 month. It uses the Bool argument to determine whether we're
 in a leap year (which should only affect the answer for
 February). By the way, I never remember that supposedly-
 mnemonic saying beyond "thirty days has September" -- I have
 to look them up all the time.
-}

numDaysIn :: Bool -> Month -> Int
numDaysIn leap month = 1

{- Almost done, but we've saved the trickiest for last! Just do
 your best, and get as close as you feel you can. But it's
 better to have a non-working (or partly working) function
 that COMPILES rather than something with compiler errors. So
 if you can't get something to compile, comment it out and I
 can take a look. That way the tests for the work above this
 point can still run!

 We want to be able to convert calendar days into an ordinal
 number, starting with ZERO to represent January 1st. These
 functions take a Boolean indicating whether we're in a leap
 year, which would affect ALL DAYS AFTER February 28th. The
 ordinals just keep counting as the days and months change, so
 30 represents January 31st, and then 31 is February 1st, 32
 is February 2nd, etc. In a leap year, December 31st will be
 365, but in a non-leap year 364.
-}

ordToCalDay :: Bool -> Int -> CalDay
ordToCalDay leap day = CalDay Jan 1

calDayToOrd :: Bool -> CalDay -> Int
calDayToOrd leap (CalDay month day) = 0

{- You don't need to do anything with the instance definition
 below, it's just showing that (once the above ordinal
 conversions are working) we can define an Enum instance (this
 one assumes it's NOT a leapyear) that lets us use ranges and
 succ/pred for calendar days:

     ghci> succ (CalDay Mar 31)
     Apr-01

     ghci> [CalDay Apr 28 .. CalDay Jun 3]
     [Apr-28,Apr-29,Apr-30,May-01,May-02,May-03,May-04,May-05,
      May-06,May-07,May-08,May-09,May-10,May-11,May-12,May-13,
      May-14,May-15,May-16,May-17,May-18,May-19,May-20,May-21,
      May-22,May-23,May-24,May-25,May-26,May-27,May-28,May-29,
      May-30,May-31,Jun-01,Jun-02,Jun-03]

-}

instance Enum CalDay where
  toEnum = ordToCalDay False
  fromEnum = calDayToOrd False

-----------------------------------------------------------------
-- Test program -- don't change anything below

main = runTests $ do
  -- Eq instance for CalDay
  assertTrue $ CalDay Jun 3 == CalDay Jun 3
  assertTrue $ CalDay Jun 3 /= CalDay Jun 4
  assertTrue $ CalDay Dec 15 /= CalDay Nov 15
  -- Ord instance for CalDay
  assertTrue $ CalDay Jun 5 < CalDay Jul 1
  assertTrue $ CalDay Dec 5 < CalDay Dec 8
  -- Show instance for CalDay
  show (CalDay Mar 15) @?= "Mar-15"
  show (CalDay Jun 8) @?= "Jun-08"  -- leading zero
  -- Bounded instance for CalDay
  minBound @?= CalDay Jan 1
  maxBound @?= CalDay Dec 31
  -- num days per month
  numDaysIn True Jun @?= 30
  numDaysIn False Jun @?= 30
  numDaysIn True Feb @?= 29
  numDaysIn False Feb @?= 28
  sum (map (numDaysIn False) [Jan ..]) @?= 365
  sum (map (numDaysIn True) [Jan ..]) @?= 366
  -- days to ordinals
  calDayToOrd False (CalDay Feb 15) @?= 45
  calDayToOrd False (CalDay Mar 27) @?= 85
  calDayToOrd False (CalDay Nov 30) @?= 333
  calDayToOrd True (CalDay Feb 15) @?= 45
  calDayToOrd True (CalDay Mar 27) @?= 86
  calDayToOrd True (CalDay Nov 30) @?= 334
  -- ordinals to days
  ordToCalDay False 46 @?= CalDay Feb 16
  ordToCalDay False 86 @?= CalDay Mar 28
  ordToCalDay False 334 @?= CalDay Dec 1
  ordToCalDay True 46 @?= CalDay Feb 16
  ordToCalDay True 86 @?= CalDay Mar 27
  ordToCalDay True 334 @?= CalDay Nov 30
  -- Enum instance (assumes non-leap-year)
  succ (CalDay Jul 4) @?= CalDay Jul 5
  succ (CalDay Jan 31) @?= CalDay Feb 1
  pred (CalDay Dec 1) @?= CalDay Nov 30
  length [minBound .. CalDay Mar 15] @?= 74

-- Test framework -- don't change anything below

infix  1 @?~, @?=

type TestM = WriterT (Sum Int) IO

runTests :: TestM () -> IO ()
runTests tests = do
  numFails <- getSum <$> execWriterT tests
  when (numFails > 0) $ exitWith (ExitFailure numFails)

(@?~) :: (HasCallStack, Fractional a, Ord a, Show a)
  => a -> a -> TestM ()
(@?~) a1 a2 =
  testBinRel callStack "~="
  (\a b -> abs (a-b) <= 0.00001) a1 a2
```

```haskell
(@?=) :: (HasCallStack, Eq a, Show a) => a -> a -> TestM ()
(@?=) a1 a2 =
  testBinRel callStack "==" (==) a1 a2

assertTrue :: HasCallStack => Bool -> TestM ()
assertTrue expr =
  testReport callStack expr "expected true"

assertFalse :: HasCallStack => Bool -> TestM ()
assertFalse expr =
  testReport callStack (not expr) "expected false"

testBinRel stack repr predicate a1 a2 =
  testReport stack (predicate a1 a2)
  (unwords [show a1, repr, show a2])

testReport stack pass descr = do
  unless pass (tell (Sum 1))
  lift . putStrLn $ unwords
    [ if pass then " OK " else "FAIL"
    , "line"
    , getLineNum stack
    , ":"
    , descr
    ]

getLineNum stack =
  case getCallStack stack of
    [] -> "?"
    (_, loc):_ -> show (srcLocStartLine loc)
```