Calc.hs

{- Name: Richard Eisenberg
  File: Calc.hs
  Desc: A simple calculator
-}

-- programs that you want to run independently of GHC
-- must be in a module named "Main" (or you can leave
-- out the module header, as the module name defaults
-- to Main)
module Main where

import Text.Read -- this gives access to readMaybe

-- If a return type of a variable is labeled with "IO", that
-- means that the variable contains an /action/. Executing
-- an action can cause side effects.
--
-- The type () is the empty tuple. It contains no information.
-- It is often called "unit". So, 'main' does some side
-- effects (like reading/writing to a terminal window) and
-- then results with no information -- just like a 'void'
-- method in Java.
--
-- The following actions are used below:
--   putStrLn :: String -> IO ()
--   putStr   :: String -> IO ()
--   getLine  :: IO String

main :: IO ()
main = do
  putStrLn "Welcome to the calculator!"
  putStr "What operation would you like to perform? (+, -, *, /) 
  op <- getLine

  putStr "First operand: 
  arg1 <- getLine

  putStr "Second operand: 
  arg2 <- getLine

  -- arg1 and arg2 are Strings. But we need to convert these
  -- Strings to numbers before we can compute. We use the function
  -- readMaybe :: Read a => String -> Maybe a
  -- This function converts a string to some other type a, as long
  -- as we know how to Read a. Of course, conversion might fail,
  -- so we get a Maybe.
  -- 'let' allows us to assign a new variable inside a do, but
  -- it won't run an IO action (like <- does)
  let m_arg1 = readMaybe arg1
  let m_arg2 = readMaybe arg2

  let result = performOperation op m_arg1 m_arg2
  putStrLn result
  putStrLn "Good-bye."

-- If all the arguments are valid, performs the operation requested.
-- Otherwise, returns an error string.
performOperation :: String          -- the operator
                  -> Maybe Integer   -- argument #1
                  -> Maybe Integer   -- argument #2
                  -> String
performOperation "+" (Just a1) (Just a2) = show (a1 + a2)
performOperation "-" (Just a1) (Just a2) = show (a1 - a2)
performOperation "*" (Just a1) (Just a2) = show (a1 * a2)
performOperation "/" (Just a1) (Just a2) = show (a1 'div' a2)
performOperation _ _ _ = "Invalid inputs"