```
--
-- Text-based user interface for address book application
--
-- Author: Gabriele Keller

module ADB_UserInterface (
  Command(..),
  mainDialog,           -- IO Command
                        -- show options of main menu,
                        -- read coomand from user
  readSearchStr,        -- IO String
                        -- read search string from user
  displayMatches,       -- [Address]  -> IO ()
                        -- display enumerated list of all
                        -- matches from search or delete use case
  readAddressDialogue,  -- IO (Address)
                        -- read address from user
  changeAddressDialogue,-- :: Address -> IO (Address)
                         -- update a given address
  readChoice            -- Int -> IO Int
                        -- given the no of matches, read delete
                        -- choice from user
) where

import AddressDB

data Command =
  AddEntry |
  Search   |
  Delete   |
  Change   |
  Quit


-- read search string from user
readSearchStr:: IO String
readSearchStr =
  do
     putStr "Enter search string: "
     getLine

-- read address data from user. Only minimal
-- error checking implemented so far.
readAddressDialogue:: IO (Address)
readAddressDialogue =
  do
    putStr "Enter first name: "
    fstName <- getLine
    putStr "Enter last name: "
    lastName <- getLine
    putStr "Enter telephone number: "
    phoneNo <- readInt
    return (Address fstName lastName phoneNo)

-- show all addresses of a list enumerated,
-- starting with '1'
displayMatches:: [Address]  -> IO ()
displayMatches  adrs =
    do
      let adrStrs = enumAddrStrings 1 adrs
      let maxInd  = (length adrs) - 1
      if adrStrs == "" then
         do
           putStr "no match found\n"
        else
          do
           putStr adrStrs
    where
      enumAddrStrings n [] = ""
```

```
      enumAddrStrings n (adr:adrs) =
        (show n) ++ "\t" ++ (showAddress adr) ++ "\n" ++
        (enumAddrStrings (n+1) adrs)

-- convert an address into a string
showAddress:: Address -> String
showAddress (Address fn ln no) =
  fn ++ " " ++ ln ++ "\t" ++ (show no)


-- read an Integer value - make sure input is non-empty
-- and only contains digits
readInt:: IO Int
readInt =
  do
    noStr <- getLine
    if (noStr == "") then
        do
          putStr "Please enter a number\n"
          readInt

      else if (and ['0' <= c && '9' >= c | c <- noStr]) then
        return (read noStr) else
          do
            putStr "Please enter only digits\n"
            readInt

-- print the main dialogue
printDialog:: IO ()
printDialog =
  do
    let str = "Enter one of the following commands: \n" ++
              "a  -- add an entry to the address book \n" ++
              "s  -- search  the address book\n" ++
              "d  -- delete entry\n" ++
              "c  -- change entry\n" ++
              "q  -- quit application \n"
    putStr str

-- read Command from user. Repeat if illegal input
readCommand:: IO Command
readCommand =
  do
    str <- getLine
    case str of
      "a" -> return AddEntry
      "s" -> return Search
      "d" -> return Delete
      "c" -> return Change
      "q" -> return Quit
      _   -> do putStr "Please enter a, s, or q\n"
                readCommand

-- given the max index of a list (e.g. 0 for a list
-- with 1 element, ask user to enter choice. Note
-- that the user enumeration starts with '1'
readChoice:: Int -> IO Int
readChoice maxInd =
  do
    putStr ("Select which address? (1 -" ++
                        (show (maxInd+1)) ++ ")\n")
    i <- readInt
    if ((0 < i) && i <= (maxInd+1)) then return (i-1)
      else do
             putStr "No such item (index out of range)!\n"
             (readChoice maxInd)


changeAddressDialogue:: Address -> IO (Address)
```

```
changeAddressDialogue (Address fn ln no) =
    do putStr "Change first name? (press return for no changes)\n"
       putStr (fn ++ "\n")
       newFn' <- getLine
       let newFn = if (newFn' == "") then fn else newFn'
       putStr "Change last name? (press return for no changes)\n"
       putStr (ln ++ "\n")
       newLn' <- getLine
       let newLn = if (newLn' == "") then ln else newLn'
       putStr "Change telephone no? (enter '0' no changes)\n"
       putStr ((show no)  ++ "\n")
       newNo' <- readInt
       let newNo = if (newNo' == 0) then  no else newNo'
       return (Address newFn newLn newNo)

-- show options and read command
mainDialog:: IO Command
mainDialog =
  do
    printDialog
    readCommand
```