```haskell
module Main where

import Grad
import System.Environment(getArgs)
import System.Exit(die)
import Data.List(isInfixOf)

{- |
Module      :  <File name or $Header$ to be replaced automatically>
Description :  Parallelized Gradient Descent algorithm for linear regression
Copyright   :  (c) <Max Helman, Riya Chakraborty>
License     :  BSD 3-Clause

Maintainer  :  mhh2148@columbia.edu, rc3242@columbia.edu
Stability   :  stable
Portability :  portable
-}

main :: IO()
main = do
        args <- getArgs
        input <- case args of
           [f, method, guess, parseq, chunks] -> return [f, method, guess, parseq, chunks]
           _ -> do
                die $ "Usage: grad-descent <filename> <loss function: linear/logistic> <guess array> <parallel/sequential>
<number of chunks>"
        csvData <- getCSVData (head input)

        let linMatch = or $ map ($ (head $ tail input)) (map isInfixOf ["Linear", "linear", "LINEAR"])
        let logMatch = or $ map ($ (head $ tail input)) (map isInfixOf ["Logistic", "logistic", "LOGISTIC"])

        appLoss <- case (linMatch || logMatch) of
            True -> if linMatch then (return computeGradRowLinear) else (return computeGradRowLogistic)
            False -> do
                    die $ "Choose either Linear or Logistic loss functions"

        let guess = read (head $ tail $ tail input) :: [Double]
        let choice = last $ init input
        let chunkNum = read $ last input :: Int

        print $ descendSteps choice chunkNum csvData appLoss guess (1000::Int) (0.0000001::Double)
--          print $ descendSteps choice chunkNum csvData appLoss guess (10000::Int) (0.001::Double)
```