```
Beginning tests for thakkarj Mon Sep 12 14:58:58 AEST 2016

Compiling sources
ghc -O2 --make studenttest
[1 of 3] Compiling Card             ( Card.hs, Card.o )
[2 of 3] Compiling Proj1            ( Proj1.hs, Proj1.o )
[3 of 3] Compiling Main             ( studenttest.hs, studenttest.o )
Linking studenttest ...

Running tests
Testing submission for thakkarj

Standard test case  1 (2 cards) ... 4 guesses
Standard test case  2 (2 cards) ... 5 guesses
Standard test case  3 (2 cards) ... 5 guesses
Standard test case  4 (2 cards) ... 4 guesses
Standard test case  5 (2 cards) ... 3 guesses
Standard test case  6 (2 cards) ... 5 guesses
Standard test case  7 (2 cards) ... 3 guesses
Standard test case  8 (2 cards) ... 4 guesses
Standard test case  9 (2 cards) ... 4 guesses
Standard test case 10 (2 cards) ... 5 guesses
Standard test case 11 (2 cards) ... 4 guesses
Standard test case 12 (2 cards) ... 4 guesses
Standard test case 13 (2 cards) ... 4 guesses
Standard test case 14 (2 cards) ... 4 guesses
Standard test case 15 (2 cards) ... 5 guesses
Standard test case 16 (2 cards) ... 5 guesses
Standard test case 17 (2 cards) ... 5 guesses
Standard test case 18 (2 cards) ... 4 guesses
Standard test case 19 (2 cards) ... 3 guesses
Standard test case 20 (2 cards) ... 3 guesses
Standard test case 21 (2 cards) ... 5 guesses
Standard test case 22 (2 cards) ... 3 guesses
Standard test case 23 (2 cards) ... 4 guesses
Standard test case 24 (2 cards) ... 3 guesses
Standard test case 25 (2 cards) ... 4 guesses
Standard test case 26 (2 cards) ... 5 guesses
Standard test case 27 (2 cards) ... 3 guesses
Standard test case 28 (2 cards) ... 4 guesses
Standard test case 29 (2 cards) ... 3 guesses
Standard test case 30 (2 cards) ... 5 guesses
    Hard test case  1 (3 cards) ... 10 guesses
    Hard test case  2 (3 cards) ... 10 guesses
    Hard test case  3 (3 cards) ... 7 guesses
    Hard test case  4 (3 cards) ... 6 guesses
    Hard test case  5 (3 cards) ... 9 guesses
    Hard test case  6 (4 cards) ... 8 guesses
    Hard test case  7 (4 cards) ... 6 guesses
    Hard test case  8 (4 cards) ... 10 guesses
    Hard test case  9 (4 cards) ... 7 guesses
    Hard test case 10 (4 cards) ... 9 guesses

       Standard tests attempted          :   30
       Standard tests passed             :   30
       Standard total guesses            :  122

           Hard tests attempted          :   10
           Hard tests passed             :   10
```

```
          Hard total guesses              :   82

Results Summary

            900 feedback tests     (/ 10) :   10

       Standard correctness points (/ 20) :   20
       Standard quality points     (/ 30) :   30

           Hard correctness points (/  5) :    5
           Hard quality points     (/  5) :    4

                       Total Points (/ 70) :   69

Completed tests Mon Sep 12 14:59:10 AEST 2016
```

```
--  File     : Proj1.hs
--  Author   : Jigar Thakkar
--  Purpose  : Guessing program for proj1 project

-- | This code implements a GameState type, initialGuess and
--   nextGuess functions. This program technically works with
--   any number of cards and gives output with the minimum number
--   of guesses.

module Proj1 (feedback, initialGuess, nextGuess, GameState) where

import Card
import Data.List

type GameState = [[Card]]

-- | Takes two cards guesses and Calculates feedback according to given two card
s,
--   This feedback contains:
--   1. correctCards: How many of the cards in the answer are also in the guess
--   2. lessGuess:    How many cards in the answer have rank lower than the lowe
st
--      rank in the guess (lower ranks).
--   3. sameGuess:    How many of the cards in the answer have the same rank as
a
--      card in the guess (correct ranks).
--   4. greterGuess:  How many cards in the answer have rank higher than the hig
hest
--      rank in the guess (higher ranks).
--   5. sameSuite:    How many of the cards in the answer have the same suit as
a
--      card in the guess.

feedback :: [Card]-> [Card] -> ( Int , Int , Int , Int , Int )
feedback  answer guess = (correctCards answer guess,lessGuess answer guess,sameG
uess answer guess,greterGuess answer guess ,sameSuite answer guess)

-- | Number of guess cards which exactly matches the answer cards
correctCards :: [Card] -> [Card] -> Int
correctCards _ [] = 0
correctCards x (y:ys)
          |(elem y x) == True = 1 + (correctCards x ys)
          |otherwise = (correctCards x ys)


-- | Checks for the minimum rank in guess cards. Then counts how many cards are
less
--   than that rank in the answer cards
lessGuess:: [Card]->[Card] -> Int
lessGuess [] _=0
lessGuess (x:xs) y
          |(rank x) < minimum (onlyRank y) = lessGuess xs y+1
          |otherwise = lessGuess xs y


-- | Checks the number of cards whose rank matches exactly amongst the guess and
 answer
sameGuess :: [Card]-> [Card] -> Int
sameGuess answer guess = length ((onlyRank answer) \\ ((onlyRank answer) \\ (onl
```

File-level documentation should describe problem domain and solution methodology

Wrapped lines compromise readability

Redundant `==True`

```
yRank guess)))


-- | Checks for the maximum rank in the guess cards. Then counts how many cards
are
--   greater than that rank in the answer cards
greterGuess::[Card]->[Card]->Int
greterGuess [] _ = 0
greterGuess (x:xs) y
            |(rank x) > maximum (onlyRank y) = greterGuess xs y + 1
            |otherwise = greterGuess xs y

-- | Helper function for lessGuess,sameGuess and greterGuess which takes card li
st
--   and seperate rank from it and gives rank list as output
onlyRank :: [Card] -> [Rank]
onlyRank [] = []
onlyRank (x:xs) = (rank x):(onlyRank xs)


-- | Checks how many answer cards matches the guess card's suit
sameSuite ::[Card]-> [Card] -> Int
sameSuite answer guess =length( (onlySuit answer) \\ ((onlySuit answer) \\ (only
Suit guess)))
```

Good use of `Data.List`

```
-- | Helper function for sameSuite which takes card list and seperate suit
--   from it and gives suit list as output
onlySuit :: [Card] -> [Suit]
onlySuit [] = []
onlySuit (x:xs) = (suit x):(onlySuit xs)


-- | initialGuess is first guess given by this program initialGuess is
--   first guess given by this program it takes number of cards in single
--   guess, for example, "2C AS" contains two card so it gives two cards as
--   initial guess and if  cards are more than two than it creates first
--   guess and game state according to that

initialGuess :: Int -> ([Card],GameState)
initialGuess n
           | n == 2    = ([Card Diamond R6, Card Club Jack], initalGameState)
           | otherwise = (head initalGameState, tail initalGameState)
                 where initalGameState = makeGameState n
```

Unexplained hard-coded first guess

```
-- | makeGameState is a GameState like a function which generates every
--   possible guesses.
makeGameState :: Int -> GameState
makeGameState n = filter duplicateFilter initalGameState
                where initalGameState = sequence (permutationCards n)

-- | permutationCards takes number of card in guess, and list of list of every
--   possible cards, for example 3 then [[2C,3C,4C]..[QS,KS,AS]] something like
--   this.
permutationCards :: Int -> [[Card]]
permutationCards 1 = [singleDeck]
permutationCards n = singleDeck:permutationCards (n-1)

-- | Contains a list of all the possible cards of singleDeck [2C..AS]
singleDeck = [minBound..maxBound] :: [Card]
```

```
-- | duplicateFilter is returen boolean value,
--   Return True when card guess is only one time in list
--   Rrturn False when list contains same guess
duplicateFilter :: Eq a => [a] -> Bool
duplicateFilter []        = True
duplicateFilter (x:xs)
         | xs == []      = True
         | x == head xs  = False
         | otherwise     = not (x `elem` xs) && duplicateFilter xs
```

Excellent description of derivation of hard-coded guesses

```
{- we can make initialGuess hardCoded like this also, if we sure that we have
   only certain number of card in single guess
initialGuess :: Int -> ([Card], GameState)
initialGuess numberOfCards
             |numberOfCards == 2 = ([Card Heart R5,Card Diamond R10], (delete [C
ard Heart R5,Card Diamond R10] [ [x ,y]| x <- singleDeck, y <- singleDeck, x < y
 ]))
             |numberOfCards == 3 = ([Card Heart R5,Card Diamond R5,Card Spade R5
], (delete [Card Heart R5,Card Diamond R5,Card Spade R5] [ [x,y,z] | x <- single
Deck, y <- singleDeck,z<-singleDeck, (x < y && y < z) ]))
             |numberOfCards == 4 = ([Card Club R5,Card Diamond R5,Card Heart R5,
Card Spade R5], (delete [Card Club R5,Card Diamond R5,Card Heart R5,Card Spade R
5] [ [x,y,z,p] | x <- singleDeck, y <- singleDeck,z<-singleDeck,p<-singleDeck, (
x < y && y < z && z < p) ]))
-}


-- | nextGuess is repeatedly giving a new guess and new GameState after the
--   initial guess according to feedback until we get the correct answer.
--   It also filters out gameState and removes unnecessary guesses from guess
--   list of cards.

nextGuess :: ([Card],GameState) -> (Int,Int,Int,Int,Int) -> ([Card],GameState)
nextGuess (initialGuess, gameState) lastFeedback = ( (head (filter (\xs->(feedba
ck xs initialGuess) == lastFeedback) gameState)) , (filter (\n->(feedback n init
ialGuess) == lastFeedback) gameState))
```

Excellent function-level documentation

Functions and variables are generally well named and easy to understand