```
Beginning tests for xiong1 Wed Sep  7 17:42:21 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 xiong1

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

       Standard tests attempted          :   30
       Standard tests passed             :   30
       Standard total guesses            :  127

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

```
           Hard total guesses              :   48

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) :    5

                        Total Points (/ 70) :   70

Completed tests Wed Sep  7 17:42:23 AEST 2016
```

```
--  File     : Proj1.hs
--  Author   : Student Name: Xiong1   StudentId: 722890
--  Purpose  : Program for project1

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

import Data.List
import Card

--  GameState type to store the list of remaining possible answers
type GameState = [[Card]]
type Feedback = (Int, Int, Int, Int, Int)


-- | Give two list of cards, which are called "target" and "guess"
--   then use five other functions to get five feedback numbers as a tuple.
feedback :: [Card] -> [Card] -> (Int, Int, Int, Int, Int)
feedback (c1:c1s) (c2:c2s) =
        (exactMatch (c1:c1s) (c2:c2s), lessRank (c1:c1s) (c2:c2s),
        rankMatch (c1:c1s) (c2:c2s), greaterRank (c1:c1s) (c2:c2s),
        suitMatch (c1:c1s) (c2:c2s))


-- | Give two lists of cards (target, guess), returns the number of cards in
--   the target and also in the guess.
exactMatch :: [Card] -> [Card] -> Int
exactMatch (c1:c1s) (c2:c2s)= length (filter (== True)
                              [ c1 == c2 | c1 <- (c1:c1s), c2 <- (c2:c2s)])


-- | Give two list of cards (target, guess), returns the number of cards in
--   the target having rank lower than the lowest rank in the guess."rank" is
--   the function used to get the rank a card.
lessRank :: [Card] -> [Card] -> Int
lessRank (c1:c1s) (c2:c2s) = length (filter (< minimum
                               (map rank (c2:c2s))) (map rank(c1:c1s)))


-- | Give two list of cards (target, guess), returns the number of cards in
--   the target have the same rank as a card in the guess. "intersection"
--   funcion on the Data.List library can not be used inthis case, because
--   this function will remove deplicate element in the list. Therefore, we
--   define our own funcitons called "intersectRank" and "intersectSuit" to
--   get the intersection part of two lists of cards.
rankMatch :: [Card] -> [Card] -> Int
rankMatch c1s c2s = length (intersectRank (map rank c1s) (map rank c2s ))


-- | Give two lists of cards (target, guess), returns the number of cards in
--   the target having rank higher than the highest rank in the guess.
greaterRank :: [Card] -> [Card] -> Int
greaterRank (c1:c1s) (c2:c2s) = length (filter (> maximum
                                  (map rank (c2:c2s))) (map rank (c1:c1s)))


-- | Give two lists of cards (target, guess), returns the number of cards in
--   the target having the same suit as a card in the guess. "suit" is the
--   function used to get the suit of a card.
suitMatch :: [Card] -> [Card] -> Int
```

```
suitMatch c1s c2s = length (intersectSuit (map suit c1s) (map suit c2s ))


-- | Give two lists of cards' suit, then returns the intersection suit and
--   keep the duplicate suit in the intersection
intersectSuit :: [Suit] -> [Suit] -> [Suit]
intersectSuit c1s c2s = c1s\\(c1s\\c2s)


-- | Give two lists of cards' rank, then returns the intersection rank and
--   keep the duplicate rank in the intersection
intersectRank :: [Rank] -> [Rank] -> [Rank]
intersectRank c1s c2s = c1s\\(c1s\\c2s)


-- | This funciton takes numbers of cards in the answer as input, then
--   then returns a list of specifid cards and a GameState sotred the list
--   of remaining possible answers, which is a list of card lists. In
--   this project, we only consider 2-4 cards cases, so iuput other number
--   of cards will get an error message. "gs2", "gs3", "gs4" means all
--   remaining possible answers for 2 cards, 3 cards and 4 cards. For
--   example, all possible answers for 2 cards are 52*51/2 = 1326, beacuse
--   we ingore the order of cards and we just assume first card is smaller
--   than second card (c1 < c2).
initialGuess :: Int -> ([Card], GameState)
initialGuess number
    | number == 2 = ( [Card Club R6, Card Club R10], gs2)
    | number == 3 = ( [Card Club R4, Card Diamond R7, Card Heart R10], gs3)
    | number == 4 = ( [Card Club R3, Card Heart R6, Card Diamond R9,
                       Card Spade Jack ], gs4)
    | otherwise = error "The cards number should be 2, 3 or 4. "
    where
      gs2 = [[c1, c2] | c1 <- allCards, c2 <- allCards, Â c1 < c2]
      gs3 = [[c1, c2, c3] | c1 <- allCards , c2 <- allCards,
            c3 <- allCards, c1 < c2 && c2 < c3 && c1 < c3]
      gs4 = [[c1, c2, c3, c4] |  c1 <- allCards, c2 <- allCards,
            c3 <- allCards, c4 <- allCards, c1 < c2 && c1< c3 &&
            c1 < c4 && c2 < c3 && c2 < c4 && c3 < c4]
      allCards = [ Card s r | s <- [Club, Diamond, Heart, Spade],
                 r <- [R2, R3, R4, R5, R6, R7, R8, R9, R10,
                 Jack, Queen, King, Ace]]


-- | This function takes a previous guess, gamestate and the feedback,
--   returns a tuple contains next gusss cards combinition and another
--   gamestate, which contains remaining possible answers.
nextGuess :: ([Card], GameState) -> (Int, Int, Int, Int, Int)
             -> ([Card], GameState)
nextGuess (guess, (c:rest)) fb = (head newGs, newGs)
    where newGs = (pareGameState (guess, (c:rest)) fb)


-- | After we get the feedback of the guess, we use it to pare the gamestate
--   step by step, which means shrink the range of remaining possible answers.
--   For each guess, we only keep the possible answers has the same feedback
--   as previous guess.
pareGameState :: ([Card], GameState) -> (Int, Int, Int, Int, Int) -> GameState
pareGameState (guess, gs) fb =
  filter (\x -> feedback x guess ==fb) gs
```