```
Compiling submission
[1 of 3] Compiling HaskellTest      ( HaskellTest.hs, HaskellTest.o )
[2 of 3] Compiling Proj1            ( Proj1.hs, Proj1.o )
[3 of 3] Compiling Main             ( studenttest.hs, studenttest.o )
Linking studenttest ...
Beginning tests at Thu 23 Apr 2020 19:54:39 AEST
Haskell test run started 2020-04-23 19:54:40.077855 AEST
Testing feedback function
                 Feedback test   1 ... PASSED
                 Feedback test   2 ... PASSED
                 Feedback test   3 ... PASSED
                 Feedback test   4 ... PASSED
                 Feedback test   5 ... PASSED
                 Feedback test   6 ... PASSED
                 Feedback test   7 ... PASSED
                 Feedback test   8 ... PASSED
                 Feedback test   9 ... PASSED
                 Feedback test  10 ... PASSED
                 Feedback test  11 ... PASSED
                 Feedback test  12 ... PASSED
                 Feedback test  13 ... PASSED
                 Feedback test  14 ... PASSED
                 Feedback test  15 ... PASSED
                 Feedback test  16 ... PASSED
                 Feedback test  17 ... PASSED
                 Feedback test  18 ... PASSED
                 Feedback test  19 ... PASSED
                 Feedback test  20 ... PASSED
                 Feedback test  21 ... PASSED
                 Feedback test  22 ... PASSED
                 Feedback test  23 ... PASSED
                 Feedback test  24 ... PASSED
                 Feedback test  25 ... PASSED
                 Feedback test  26 ... PASSED
                 Feedback test  27 ... PASSED
                 Feedback test  28 ... PASSED
                 Feedback test  29 ... PASSED
                 Feedback test  30 ... PASSED
                 Feedback test  31 ... PASSED
                 Feedback test  32 ... PASSED
                 Feedback test  33 ... PASSED
                 Feedback test  34 ... PASSED
                 Feedback test  35 ... PASSED
                 Feedback test  36 ... PASSED
                 Feedback test  37 ... PASSED
                 Feedback test  38 ... PASSED
                 Feedback test  39 ... PASSED
                 Feedback test  40 ... PASSED
                 Feedback test  41 ... PASSED
                 Feedback test  42 ... PASSED
                 Feedback test  43 ... PASSED
                 Feedback test  44 ... PASSED
                 Feedback test  45 ... PASSED
                 Feedback test  46 ... PASSED
                 Feedback test  47 ... PASSED
                 Feedback test  48 ... PASSED
                 Feedback test  49 ... PASSED
                 Feedback test  50 ... PASSED
                 Feedback test  51 ... PASSED
```

```
Feedback test  52 ... PASSED
Feedback test  53 ... PASSED
Feedback test  54 ... PASSED
Feedback test  55 ... PASSED
Feedback test  56 ... PASSED
Feedback test  57 ... PASSED
Feedback test  58 ... PASSED
Feedback test  59 ... PASSED
Feedback test  60 ... PASSED
Feedback test  61 ... PASSED
Feedback test  62 ... PASSED
Feedback test  63 ... PASSED
Feedback test  64 ... PASSED
Feedback test  65 ... PASSED
Feedback test  66 ... PASSED
Feedback test  67 ... PASSED
Feedback test  68 ... PASSED
Feedback test  69 ... PASSED
Feedback test  70 ... PASSED
Feedback test  71 ... PASSED
Feedback test  72 ... PASSED
Feedback test  73 ... PASSED
Feedback test  74 ... PASSED
Feedback test  75 ... PASSED
Feedback test  76 ... PASSED
Feedback test  77 ... PASSED
Feedback test  78 ... PASSED
Feedback test  79 ... PASSED
Feedback test  80 ... PASSED
Feedback test  81 ... PASSED
Feedback test  82 ... PASSED
Feedback test  83 ... PASSED
Feedback test  84 ... PASSED
Feedback test  85 ... PASSED
Feedback test  86 ... PASSED
Feedback test  87 ... PASSED
Feedback test  88 ... PASSED
Feedback test  89 ... PASSED
Feedback test  90 ... PASSED
Feedback test  91 ... PASSED
Feedback test  92 ... PASSED
Feedback test  93 ... PASSED
Feedback test  94 ... PASSED
Feedback test  95 ... PASSED
Feedback test  96 ... PASSED
Feedback test  97 ... PASSED
Feedback test  98 ... PASSED
Feedback test  99 ... PASSED
Feedback test 100 ... PASSED
Feedback test 101 ... PASSED
Feedback test 102 ... PASSED
Feedback test 103 ... PASSED
Feedback test 104 ... PASSED
Feedback test 105 ... PASSED
Feedback test 106 ... PASSED
Feedback test 107 ... PASSED
Feedback test 108 ... PASSED
Feedback test 109 ... PASSED
Feedback test 110 ... PASSED
```

```
                Feedback test 111 ... PASSED
                Feedback test 112 ... PASSED
                Feedback test 113 ... PASSED
                Feedback test 114 ... PASSED
                Feedback test 115 ... PASSED
                Feedback test 116 ... PASSED
                Feedback test 117 ... PASSED
                Feedback test 118 ... PASSED
                Feedback test 119 ... PASSED
                Feedback test 120 ... PASSED
Available points: 20.0
Points earned: 20.0

Testing guessing functions
                   Guess test   1 ... PASSED 6.0
                   Guess test   2 ... PASSED 6.0
                   Guess test   3 ... PASSED 7.0
                   Guess test   4 ... PASSED 5.0
                   Guess test   5 ... PASSED 7.0
                   Guess test   6 ... PASSED 4.0
                   Guess test   7 ... PASSED 4.0
                   Guess test   8 ... PASSED 4.0
                   Guess test   9 ... PASSED 8.0
                   Guess test  10 ... PASSED 6.0
                   Guess test  11 ... PASSED 6.0
                   Guess test  12 ... PASSED 8.0
                   Guess test  13 ... PASSED 5.0
                   Guess test  14 ... PASSED 9.0
                   Guess test  15 ... PASSED 6.0
                   Guess test  16 ... PASSED 6.0
                   Guess test  17 ... PASSED 5.0
                   Guess test  18 ... PASSED 12.0
                   Guess test  19 ... PASSED 5.0
                   Guess test  20 ... PASSED 4.0
                   Guess test  21 ... PASSED 7.0
                   Guess test  22 ... PASSED 10.0
                   Guess test  23 ... PASSED 9.0
                   Guess test  24 ... PASSED 5.0
                   Guess test  25 ... PASSED 5.0
                   Guess test  26 ... PASSED 6.0
                   Guess test  27 ... PASSED 8.0
                   Guess test  28 ... PASSED 3.0
                   Guess test  29 ... PASSED 12.0
                   Guess test  30 ... PASSED 4.0
                   Guess test  31 ... PASSED 6.0
                   Guess test  32 ... PASSED 8.0
                   Guess test  33 ... PASSED 7.0
                   Guess test  34 ... PASSED 10.0
                   Guess test  35 ... PASSED 8.0
                   Guess test  36 ... PASSED 6.0
                   Guess test  37 ... PASSED 8.0
                   Guess test  38 ... PASSED 7.0
                   Guess test  39 ... PASSED 6.0
                   Guess test  40 ... PASSED 4.0
Total tests: 40.0
Tests successfully guessed: 40.0
Total guesses for successful tests: 262.0
Average guesses: 6.55
Available points: 50.0
```

```
Points earned: 49.347176287141146

Overall Results:
Available points: 70.0
Points earned: 69.34717628714114
Haskell test run ended 2020-04-23 19:54:52.799389 AEST
Total CPU time used = 12720 milliseconds
Completed tests at Thu 23 Apr 2020 19:54:52 AEST
```

```
{-------------------------------------------------------------------------------
-- Author: Zexi Liu
-- Student Number: 813212
-- Email: zexil1@student.unimelb.edu.au

-- the Implementation of Project 1 in COMP90048 Declarative Programming, 2020 S-
emester1.

-- Proj1 is a simple two-player logical guessing game created for this project.
for details please visit https://github.com/xIa066/Hide-SeekGame-Haskell

    1. The game depicts two players, searcher and hider. This source code imple-
    mented all posible actions/function that two players should have. However,
    the actual game-play simulation is not implemented in this file.

    The game-play simulation should use the functions in this file as an inter-
    face.

    2. In addition, this file constructs data structure to abstract the Game
-------------------------------------------------------------------------------}

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

------------------------------Data Structure---------------------------------
-- import other module
import Data.Char
import Data.List

--Location
type Row = Int
type Column = Char
data Location = Location Row Column
    deriving (Eq,Ord)

--Show Location
showlocation :: Location -> String
showlocation (Location r c) = [toUpper c] ++ (show r)
instance Show Location where show = showlocation

-- GameState
-- GameState is a set of consistent guesses. A guess is 3-tuple Location.
-- consistency is defined as the choice of guesses should consider the
-- previous feedback from the hider.
-- To be more specific. In the set of consistent guesses. If you call
-- feedback function, given previous guess and each possible guesses in set,
-- then the output should be the same as the answer that hider gives
-- you for your previous guess.
type GameState = [[Location]]

-- All Possible Locations
allLocations :: [Location]
allLocations = [Location 1 'A', Location 1 'B', Location 1 'C',
                        Location 1 'D', Location 1 'E', Location 1 'F',
                        Location 1 'G', Location 1 'H',
                        Location 2 'A', Location 2 'B', Location 2 'C',
                        Location 2 'D', Location 2 'E', Location 2 'F',
                        Location 2 'G', Location 2 'H',
                        Location 3 'A', Location 3 'B', Location 3 'C',
```

```
                        Location 3 'D', Location 3 'E', Location 3 'F',
                        Location 3 'G', Location 3 'H',
                        Location 4 'A', Location 4 'B', Location 4 'C',
                        Location 4 'D', Location 4 'E', Location 4 'F',
                        Location 4 'G', Location 4 'H']

-- We later [combination 3 allLocation] to represent possible guesses
-- This Combination function is written by Prof. Peter Schachte
-- for detail explanation please visit the following
-- https://mail.haskell.org/pipermail/beginners/2011-November/008991.html
combinations :: Int -> [a] -> [[a]]
combinations 0 _ = [[]]
combinations _ [] = []
combinations n xs@(y:ys)
  | n < 0     = []
  | otherwise = case drop (n-1) xs of
                  [ ] -> []
                  [_] -> [xs]
                  _   -> [y:c | c <- combinations (n-1) ys]
                         ++ combinations n ys
-------------------------------------------------------------------------------

------------------------Must-have Interface functions-------------------------

-- ***********************************
toLocation :: String -> Maybe Location
{- ***********************************
gives Just the Location named by the string, or Nothing if
the string is not a valid location name.
***********************************-}
toLocation s
  | (length s) /= 2 = Nothing -- makes sure the string length is 2
  | (column >='A'&& column <='H'&& row >= '1' && row <= '4')  -- confine scope
    = Just (Location (digitToInt row) column)
  | otherwise = Nothing
 where column = (s!!0)  -- 0 is the index of input string: Row
       row = (s!!1) -- 1 is the index of input string: Column

-- ***********************************
feedback :: [Location] -> [Location] -> (Int,Int,Int)
-- ***********************************
feedback ts gs
  | length(ts) /= 3 && length(gs) /= 3 = error "Not enough input locations"
  | otherwise = go1_fb (ts!!0, ts!!1, ts!!2) (gs!!0, gs!!1, gs!!2)
-- ts and gs are respectively a list of Three Locations. 0,1,2 are the index of
-- the list. :t ts!!0 = Location

---------------------- Helper functions: feedback -----------------------------

-- ***********************************
go1_fb :: (Location, Location, Location)
          -> (Location, Location, Location)
          -> (Int,Int,Int)
-- ***********************************
-- go1_fb compute the (Int, Int, Int) feedback value
go1_fb (Location r_t1 c_t1, Location r_t2 c_t2, Location r_t3 c_t3)
       (Location r_g1 c_g1, Location r_g2 c_g2, Location r_g3 c_g3)
      -- :t feedback_set is (Int, Int, Int). feedback_set!!i is Int, i<-0,1,2
    = (feedback_set!!0, feedback_set!!1, feedback_set!!2)
```

```
      where
        -- compute the shortest distances from target to guess

        -- for Guess Location 1 to all three Targets
         dist_t1 = minimum [cal_dsts (Location r_t1 c_t1) (Location r_g1 c_g1),
                            cal_dsts (Location r_t2 c_t2) (Location r_g1 c_g1),
                            cal_dsts (Location r_t3 c_t3) (Location r_g1 c_g1)]
        -- for Guess Location 2 to all three Targets
         dist_t2 = minimum [cal_dsts (Location r_t1 c_t1) (Location r_g2 c_g2),
                            cal_dsts (Location r_t2 c_t2) (Location r_g2 c_g2),
                            cal_dsts (Location r_t3 c_t3) (Location r_g2 c_g2)]
        -- for Guess Location 1 to all three Targets
         dist_t3 = minimum [cal_dsts (Location r_t1 c_t1) (Location r_g3 c_g3),
                            cal_dsts (Location r_t2 c_t2) (Location r_g3 c_g3),
                            cal_dsts (Location r_t3 c_t3) (Location r_g3 c_g3)]
        -- take three distances for g1,g2,g3.
        -- summarize/compute and output the feedback values
         feedback_set = compute_feedback [dist_t1, dist_t2, dist_t3]

-- ***********************************
cal_dsts :: Location -> Location -> Int
-- ***********************************
-- calculate the distance between one target location and guess location
cal_dsts (Location r_t c_t) (Location r_g c_g)
    -- the distance is to take the maximum of their coordinate difference
    = max row_diff column_diff
      where row_diff = abs (r_t - r_g)
            column_diff = abs ((ord c_t)-(ord c_g))

-- ***********************************
compute_feedback:: [Int]->[Int]
-- ***********************************
-- @input: this function takes a list of distances between target and distances
-- @return: (number of 0 dist, number of 1 dist, number of 2 dist)
-- this function also consider the possible distance that is greater than 2
compute_feedback xs = go (group (sort xs)) 0
            where go :: [[Int]]->Int->[Int]
                  go _ 3 = []
                  go [] n = 0 : go [] (n+1)
                  go (x:xs) n
                     | (x!!0)/=n = 0 : go (x:xs) (n+1)
                     | (x!!0)==n = length(x) : go xs (n+1)

-- ########################################################################## --

-- ***********************************
initialGuess :: ([Location],GameState)
-- ***********************************
initialGuess = do
    let  lx = allLocations -- see data structure session
         allGameState = combinations 3 lx -- see data structure session

         -- this hardCode is computed outside of the source code file
         -- The idea is the take the first step that return the smallest
         -- expected value.
         -- for more details please view document in function
         -- expectScore :: [Location] -> GameState -> Double.
         -- There are multiple minimums because of the semetry.
         hardCode = [[Location 1 'A', Location 1 'B', Location 2 'D']]
```

```
        -- filter out the hardCode in all possible remaining guesses
        remainGuesses = filter (\x -> not (elem x hardCode)) allGameState

    (hardCode!!0, remainGuesses)

-- *********************************
nextGuess :: ([Location],GameState) -> (Int,Int,Int) -> ([Location],GameState)
-- *********************************
nextGuess (prevGuess, prevGameState) prevFeedback = (newGuess, newGameState)
    where
          newGameState
          -- checking if all possible guesses in the set of remaining guess
          -- is consistent with previous submitted guess.
            = filter (\target -> feedback target prevGuess == prevFeedback)
                prevGameState

          (_ , newGuess)
           =
            ( -- compute a list of expected value for each remainig possible
              -- guesses.
              -- take one of the remaining guesses that has smallest
              -- expectations.
              -- minimum will work because lexical order
              minimum
                (map (\x -> (expectScore x newGameState, x)) newGameState)
            )

-- ################### Helper functions: nextGuess ##########################--
expectScore :: [Location] -> GameState -> Double
expectScore target gamestate = do
    let
        -- calculate all the possible feedbacks of the remaining locations
        allPossibleFeedback = map (\guess -> feedback target guess)
                                  gamestate
        -- group and count the number of distinct feedbacks and put it in a set
        symmetrySet = map (\x -> fromIntegral (length x)) distinct
            where distinct = group (sort allPossibleFeedback)
    -- analytical expectation formula
    -- for more details of intuition please visit
    -- https://github.com/xIa066/Hide-SeekGame-Haskell for project spec
    sum (map (\x -> x * x/ (fromIntegral (length symmetrySet))) symmetrySet)
-- ######################################################################## --
------------------------------------------------------------------------------
```