```
Haskell test run started Fri Sep 11 14:52:46 AEST 2015
Proj1 testing
                          Test   1 ... PASSED 5.0
                          Test   2 ... PASSED 5.0
                          Test   3 ... PASSED 5.0
                          Test   4 ... PASSED 5.0
                          Test   5 ... PASSED 5.0
                          Test   6 ... PASSED 6.0
                          Test   7 ... PASSED 3.0
                          Test   8 ... PASSED 6.0
                          Test   9 ... PASSED 5.0
                          Test  10 ... PASSED 5.0
                          Test  11 ... PASSED 5.0
                          Test  12 ... PASSED 4.0
                          Test  13 ... PASSED 3.0
                          Test  14 ... PASSED 3.0
                          Test  15 ... PASSED 4.0
                          Test  16 ... PASSED 5.0
                          Test  17 ... PASSED 5.0
                          Test  18 ... PASSED 5.0
                          Test  19 ... PASSED 4.0
                          Test  20 ... PASSED 6.0
                          Test  21 ... PASSED 4.0
                          Test  22 ... PASSED 3.0
                          Test  23 ... PASSED 6.0
                          Test  24 ... PASSED 5.0
                          Test  25 ... PASSED 3.0
                          Test  26 ... PASSED 6.0
                          Test  27 ... PASSED 6.0
                          Test  28 ... PASSED 4.0
                          Test  29 ... PASSED 5.0
                          Test  30 ... PASSED 5.0
                          Test  31 ... PASSED 4.0
                          Test  32 ... PASSED 7.0
                          Test  33 ... PASSED 5.0
                          Test  34 ... PASSED 5.0
                          Test  35 ... PASSED 4.0
                          Test  36 ... PASSED 4.0
                          Test  37 ... PASSED 3.0
                          Test  38 ... PASSED 4.0
                          Test  39 ... PASSED 5.0
                          Test  40 ... PASSED 4.0
                          Test  41 ... PASSED 4.0
                          Test  42 ... PASSED 4.0
                          Test  43 ... PASSED 5.0
                          Test  44 ... PASSED 4.0
                          Test  45 ... PASSED 4.0
                          Test  46 ... PASSED 6.0
                          Test  47 ... PASSED 5.0
                          Test  48 ... PASSED 4.0
                          Test  49 ... PASSED 3.0
                          Test  50 ... PASSED 5.0
                          Test  51 ... PASSED 5.0
                          Test  52 ... PASSED 4.0
                          Test  53 ... PASSED 4.0
                          Test  54 ... PASSED 4.0
                          Test  55 ... PASSED 5.0
                          Test  56 ... PASSED 5.0
                          Test  57 ... PASSED 5.0
```

```
Test  58 ... PASSED 5.0
Test  59 ... PASSED 6.0
Test  60 ... PASSED 4.0
Test  61 ... PASSED 6.0
Test  62 ... PASSED 4.0
Test  63 ... PASSED 5.0
Test  64 ... PASSED 3.0
Test  65 ... PASSED 3.0
Test  66 ... PASSED 5.0
Test  67 ... PASSED 6.0
Test  68 ... PASSED 3.0
Test  69 ... PASSED 6.0
Test  70 ... PASSED 6.0
Test  71 ... PASSED 3.0
Test  72 ... PASSED 4.0
Test  73 ... PASSED 4.0
Test  74 ... PASSED 4.0
Test  75 ... PASSED 4.0
Test  76 ... PASSED 3.0
Test  77 ... PASSED 4.0
Test  78 ... PASSED 4.0
Test  79 ... PASSED 6.0
Test  80 ... PASSED 4.0
Test  81 ... PASSED 4.0
Test  82 ... PASSED 5.0
Test  83 ... PASSED 4.0
Test  84 ... PASSED 3.0
Test  85 ... PASSED 5.0
Test  86 ... PASSED 3.0
Test  87 ... PASSED 5.0
Test  88 ... PASSED 5.0
Test  89 ... PASSED 4.0
Test  90 ... PASSED 6.0
Test  91 ... PASSED 5.0
Test  92 ... PASSED 4.0
Test  93 ... PASSED 6.0
Test  94 ... PASSED 3.0
Test  95 ... PASSED 4.0
Test  96 ... PASSED 5.0
Test  97 ... PASSED 4.0
Test  98 ... PASSED 4.0
Test  99 ... PASSED 4.0
Test 100 ... PASSED 5.0
Test 101 ... PASSED 4.0
Test 102 ... PASSED 4.0
Test 103 ... PASSED 5.0
Test 104 ... PASSED 6.0
Test 105 ... PASSED 3.0
Test 106 ... PASSED 5.0
Test 107 ... PASSED 5.0
Test 108 ... PASSED 4.0
Test 109 ... PASSED 5.0
Test 110 ... PASSED 4.0
Test 111 ... PASSED 6.0
Test 112 ... PASSED 4.0
Test 113 ... PASSED 3.0
Test 114 ... PASSED 4.0
Test 115 ... PASSED 6.0
Test 116 ... PASSED 5.0
```

```
                          Test 117 ... PASSED 4.0
                          Test 118 ... PASSED 5.0
                          Test 119 ... PASSED 5.0
                          Test 120 ... PASSED 5.0
Average guesses: 4.533333333333333
Points: 69.63411192533873 / 70.0
Haskell test run ended Fri Sep 11 14:52:47 AEST 2015
Total CPU time used = 379 milliseconds
```

```
--  File        : Proj1.hs
--  Size        : 7 KB
--  Stu_id      : 612840
--  Author      : Yash Narwal
--  Purpose     : Project 1 (Declarative Programming)
--  Modified    : 2015-09-10 18:07:05
--  Uid         : (23373/ ynarwal)
--  Gid         : (3000/ student)

-------------------------------------------------------------------------------

{-

  Introduction to the project: The logic of the program is to play two player
  guessing game. one player will select the pitche and other will try to
  guess it with minimum number of guesses needed to be made following the
  feedback got from previous guess.
  This game is easy to play but the algorithm behind can be very computational
  demanding for the best average of the guesses.

-}

-------------------------------------------------------------------------------


module Proj1 (initialGuess, nextGuess, GameState) where

import Data.List


--I am using this gamestate to keep track of previous filtered list
type GameState = [[[Char]]]
-- compare type is used for better notation and abstraction
data CompareType = Pitche | Note | Octave
                   deriving (Show, Eq, Ord)

-------------------------------------------------------------------------------

--Some Constant to define
--getting 2 as an int
indexForGuess :: Int
indexForGuess = 2

--I have run my program with all possible guesses and comes that this
--first guess gives me best average
getBestFirstGuess :: [String]
getBestFirstGuess = ["A2", "B2", "G3"]

--Layout of the game
getNumList :: [Int]
getNumList = [1,2,3]

getStringList :: [String]
getStringList = ["A","B","C", "D","E", "F", "G"]

noteIndex :: Int
noteIndex = 0

octaveIndex :: Int
```

```
octaveIndex = 1


-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-- | Computer the first guess, I find out this guess with runnning
--all possible options for the first guess and this turns out the best one
initialGuess :: ([String], GameState)
initialGuess = (getBestFirstGuess, getAllOptions)

-------------------------------------------------------------------------------



-------------------------------------------------------------------------------
{-this function calculate next guess and and a game state
in game state it returns the possible options left from all of them
Arguemnts:: (first argument list it takes a tuple of list of string which is
previous guess anda game state , in 2d argument it takes a  tuple of three
ints which are decided factors of next guess-}
nextGuess :: ([String], GameState) -> (Int, Int, Int) -> ([String], GameState)
nextGuess (target, gameState) (p,n,o) = (myGuess, newGameState)
  where myGuess  = sortedList !! index
        newGameState = filter' gameState target [p,n, o]
        index = quot (length newGameState) indexForGuess
        sortedList = quickSortReverse newGameState
-------------------------------------------------------------------------------



-------------------------------------------------------------------------------

---this function does reverse sorting on a list
quickSortReverse :: (Ord a) => [a] -> [a]
quickSortReverse [] = []
quickSortReverse (x:xs) =
  let smallerSorted = quickSortReverse [a | a <- xs, a <= x]
      biggerSorted = quickSortReverse [a | a <- xs, a > x]
  in  biggerSorted ++ [x] ++ smallerSorted
-------------------------------------------------------------------------------



-------------------------------------------------------------------------------

--This functions gives all possible options for the targets
getAllOptions =  [ [x,y,z] | x <- combination, y <-combination ,
                 z <- combination , x<y, y < z]
  where nums = getNumList
        strs = getStringList
        combination = [y ++ (show x) | x <- nums, y <- strs]
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
--Filter the list down where all remaing elements have same comparsion score
-- with the guess and thus it cut down the list, and we get a list of equal poss
ibilities items.
filter' [] (y:ys) intNumbers = []
filter' (x:xs) list2 intNumbers
  | intNumbers == getScore x list2 = x:(filter' xs list2 intNumbers)
  | otherwise                      = filter' xs list2 intNumbers
```

```
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

--This function equate if two lists have same value for a note and octave
eqauteNthElementOfList :: Eq a => CompareType -> [a] -> [a] -> Bool
eqauteNthElementOfList compareType l1 l2
  | compareType == Note = (l1 !! noteIndex) == (l2 !! noteIndex)
  | compareType == Octave = (l1 !! octaveIndex) == (l2 !! octaveIndex)

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

--we get a tuple of three ints which tell,  how much similarity,
--these two list of strings are we get three Ints for pitche,
--note and octave respectively.
getScoreTuple :: [String] -> [String] -> (Int,Int,Int)
getScoreTuple target guess = (pc, nc, oc)
  where pc =   countCorrect Pitche guess target
        nc =   countCorrect Note guess target
        oc =   countCorrect Octave guess target


--This take the tuple and chuck into a list
getScore :: [String] -> [String] -> [Int]
getScore list1 list2 = [a, b,c]
  where (a,b,c) = getScoreTuple list1 list2
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
--this functions takes a type to compate and two string's list ,
--it calculate the count for each pitche , note and octave of two given list,
-- which indeed Accordind to spec: number of correct pitche, number of correct
--note but incorrect octave and last number of crrect octave but incoorect note.
countCorrect :: CompareType -> [String] -> [String] -> Int

countCorrect compareType list1 list2
  | compareType == Pitche = pc
  | compareType == Note = nc
  | compareType == Octave = oc
  where pc = length (getListCommonElement ulist1 list2)
        tc = length ulist1
        nc = tc - ln -pc
        oc = tc - lo -pc
        ulist1 = removeDuplicates list1
        ln = length(deleteFirstsBy(eqauteNthElementOfList Note)ulist1 list2)
        lo = length(deleteFirstsBy(eqauteNthElementOfList Octave)ulist1 list2)

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

--It removes the duplicate element from a list
removeDuplicates :: Eq a => [a] -> [a]
removeDuplicates = helper []
    where helper seen1 [] = seen1
```

```
          helper seen1 (x:xs)
              | x `elem` seen1 = helper seen1 xs
              | otherwise = helper (seen1 ++ [x]) xs

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--get list of common elements of two lists
getListCommonElement :: Eq t => [t] -> [t] -> [t]


getListCommonElement [] _ = []
getListCommonElement _ [] = []

getListCommonElement list1 list2 =
  if length list1 >= length list2 then commonElementHelper ulist2 ulist1
  else commonElementHelper ulist1 ulist2
  where ulist2 = removeDuplicates list2
        ulist1 = removeDuplicates list1

commonElementHelper :: Eq t => [t] -> [t] -> [t]

commonElementHelper _ [] = []
commonElementHelper [] _ = []

commonElementHelper (x:xs) list =
  if x `elem` list then x:commonElementHelper xs list
  else commonElementHelper xs list
--------------------------------------------------------------------------------

------------------------------End of FIle---------------------------------------
```