```
Haskell test run started Mon Sep  4 19:45:07 AEST 2017
Proj1 testing
                          Test   1 ... PASSED 3.0
                          Test   2 ... PASSED 2.0
                          Test   3 ... PASSED 5.0
                          Test   4 ... PASSED 4.0
                          Test   5 ... PASSED 4.0
                          Test   6 ... PASSED 5.0
                          Test   7 ... PASSED 5.0
                          Test   8 ... PASSED 4.0
                          Test   9 ... PASSED 7.0
                          Test  10 ... PASSED 7.0
                          Test  11 ... PASSED 5.0
                          Test  12 ... PASSED 5.0
                          Test  13 ... PASSED 5.0
                          Test  14 ... PASSED 5.0
                          Test  15 ... PASSED 4.0
                          Test  16 ... PASSED 4.0
                          Test  17 ... PASSED 5.0
                          Test  18 ... PASSED 2.0
                          Test  19 ... PASSED 4.0
                          Test  20 ... PASSED 5.0
                          Test  21 ... PASSED 4.0
                          Test  22 ... PASSED 3.0
                          Test  23 ... PASSED 5.0
                          Test  24 ... PASSED 5.0
                          Test  25 ... PASSED 5.0
                          Test  26 ... PASSED 3.0
                          Test  27 ... PASSED 4.0
                          Test  28 ... PASSED 3.0
                          Test  29 ... PASSED 3.0
                          Test  30 ... PASSED 4.0
                          Test  31 ... PASSED 5.0
                          Test  32 ... PASSED 4.0
                          Test  33 ... PASSED 5.0
                          Test  34 ... PASSED 5.0
                          Test  35 ... PASSED 5.0
                          Test  36 ... PASSED 3.0
                          Test  37 ... PASSED 6.0
                          Test  38 ... PASSED 5.0
                          Test  39 ... PASSED 4.0
                          Test  40 ... PASSED 5.0
                          Test  41 ... PASSED 4.0
                          Test  42 ... PASSED 5.0
                          Test  43 ... PASSED 5.0
                          Test  44 ... PASSED 5.0
                          Test  45 ... PASSED 6.0
                          Test  46 ... PASSED 6.0
                          Test  47 ... PASSED 3.0
                          Test  48 ... PASSED 5.0
                          Test  49 ... PASSED 3.0
                          Test  50 ... PASSED 5.0
                          Test  51 ... PASSED 4.0
                          Test  52 ... PASSED 8.0
                          Test  53 ... PASSED 7.0
                          Test  54 ... PASSED 7.0
                          Test  55 ... PASSED 7.0
                          Test  56 ... PASSED 4.0
                          Test  57 ... PASSED 5.0
Test  58 ... PASSED 4.0
Test  59 ... PASSED 6.0
Test  60 ... PASSED 4.0
Test  61 ... PASSED 3.0
Test  62 ... PASSED 5.0
Test  63 ... PASSED 3.0
Test  64 ... PASSED 7.0
Test  65 ... PASSED 7.0
Test  66 ... PASSED 3.0
Test  67 ... PASSED 6.0
Test  68 ... PASSED 5.0
Test  69 ... PASSED 5.0
Test  70 ... PASSED 5.0
Test  71 ... PASSED 6.0
Test  72 ... PASSED 6.0
Test  73 ... PASSED 5.0
Test  74 ... PASSED 4.0
Test  75 ... PASSED 4.0
Test  76 ... PASSED 4.0
Test  77 ... PASSED 5.0
Test  78 ... PASSED 5.0
Test  79 ... PASSED 5.0
Test  80 ... PASSED 5.0
Test  81 ... PASSED 4.0
Test  82 ... PASSED 6.0
Test  83 ... PASSED 4.0
Test  84 ... PASSED 5.0
Test  85 ... PASSED 5.0
Test  86 ... PASSED 5.0
Test  87 ... PASSED 6.0
Test  88 ... PASSED 3.0
Test  89 ... PASSED 6.0
Test  90 ... PASSED 3.0
Test  91 ... PASSED 4.0
Test  92 ... PASSED 5.0
Test  93 ... PASSED 6.0
Test  94 ... PASSED 8.0
Test  95 ... PASSED 5.0
Test  96 ... PASSED 6.0
Test  97 ... PASSED 4.0
Test  98 ... PASSED 6.0
Test  99 ... PASSED 5.0
Test 100 ... PASSED 5.0
Test 101 ... PASSED 2.0
Test 102 ... PASSED 4.0
Test 103 ... PASSED 4.0
Test 104 ... PASSED 5.0
Test 105 ... PASSED 5.0
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 5.0
Test 113 ... PASSED 4.0
Test 114 ... PASSED 4.0
Test 115 ... PASSED 4.0
Test 116 ... PASSED 4.0
```

```
                          Test 117 ... PASSED 5.0
                          Test 118 ... PASSED 4.0
                          Test 119 ... PASSED 5.0
                          Test 120 ... PASSED 5.0
Total tests: 120.0
Tests successfully guessed: 120.0
Total guesses for successful tests: 567.0
Average guesses: 4.725
Points available: 70.0 * 120.0 / 120.0 = 70.0
Points: 69.12657617739734 / 70.0
Haskell test run ended Mon Sep  4 19:45:07 AEST 2017
Total CPU time used = 462 milliseconds
```

```
-- File: Proj1.hs
-- Author: Yitong Chen
-- Purpose: performer program for the ChordProbe game

module Proj1 (initialGuess, nextGuess, GameState) where

-- | The 'GameState' type is used to keep track of the current game state.
-- It contains a list of all the guesses left to make.

type GameState = [[String]]


-- | The 'initialGuess' function makes a initial guess A1, B1, C2

initialGuess :: ([String],GameState)
initialGuess = (["A1","B1","C2"], initialGameState)


-- | The 'nextGuess' function processes the result of previous guess, removes
-- impossible guesses from GameState, and return the next guess

nextGuess:: ([String],GameState) -> (Int,Int,Int) -> ([String],GameState)
nextGuess (guess, gameState)
        (correctPitch,correctNote,correctOctave) =
        (head newGameState, tail newGameState)
        where
                -- Filters
                -- Ensures the number of correctPitch is consistent
                filtered1 = filter (correctFilter correctPitch guess) gameState
                -- Ensures the number of right note is consistent
                filtered2 = filter (noteFilter correctNote guess) filtered1
                -- Ensures the number of right octave is consistent
                newGameState = filter (octaveFilter correctOctave guess) filtere
d2


-- | The 'initialGameState' function generates a GameState containing every
-- possible guess, filtering the duplication

initialGameState :: GameState
initialGameState = filter dupFilter buildGameState
        where
        buildGameState = sequence (buildSequence 3)


-- | The 'buildSequence' function constrcts a list of 3 lists, which contains ev
ery--  possible guesses.

buildSequence :: Int -> [[String]]
buildSequence 1 = [allPitches]
buildSequence n = allPitches:buildSequence (n-1)


-- | 'allPitches' Contains a list of all the possible pitches.

allPitches = ["A1","A2","A3","B1","B2","B3","C1","C2","C3",
        "D1","D2","D3","E1","E2","E3","F1","F2","F3","G1","G2","G3"]
```

```
-- |  The ’pitchTuple’ creates a tupple about given notes

pitchTuple :: String -> (Int, Int, Int, Int, Int, Int, Int, Int, Int, Int)
pitchTuple x
        | (x!!0) == ’A’ && (x!!1) == ’1’  = (1,0,0,0,0,0,0,1,0,0)
        | (x!!0) == ’A’ && (x!!1) == ’2’  = (1,0,0,0,0,0,0,0,1,0)
        | (x!!0) == ’A’ && (x!!1) == ’3’  = (1,0,0,0,0,0,0,0,0,1)
        | (x!!0) == ’B’ && (x!!1) == ’1’  = (0,1,0,0,0,0,0,1,0,0)
        | (x!!0) == ’B’ && (x!!1) == ’2’  = (0,1,0,0,0,0,0,0,1,0)
        | (x!!0) == ’B’ && (x!!1) == ’3’  = (0,1,0,0,0,0,0,0,0,1)
        | (x!!0) == ’C’ && (x!!1) == ’1’  = (0,0,1,0,0,0,0,1,0,0)
        | (x!!0) == ’C’ && (x!!1) == ’2’  = (0,0,1,0,0,0,0,0,1,0)
        | (x!!0) == ’C’ && (x!!1) == ’3’  = (0,0,1,0,0,0,0,0,0,1)
        | (x!!0) == ’D’ && (x!!1) == ’1’  = (0,0,0,1,0,0,0,1,0,0)
        | (x!!0) == ’D’ && (x!!1) == ’2’  = (0,0,0,1,0,0,0,0,1,0)
        | (x!!0) == ’D’ && (x!!1) == ’3’  = (0,0,0,1,0,0,0,0,0,1)
        | (x!!0) == ’E’ && (x!!1) == ’1’  = (0,0,0,0,1,0,0,1,0,0)
        | (x!!0) == ’E’ && (x!!1) == ’2’  = (0,0,0,0,1,0,0,0,1,0)
        | (x!!0) == ’E’ && (x!!1) == ’3’  = (0,0,0,0,1,0,0,0,0,1)
        | (x!!0) == ’F’ && (x!!1) == ’1’  = (0,0,0,0,0,1,0,1,0,0)
        | (x!!0) == ’F’ && (x!!1) == ’2’  = (0,0,0,0,0,1,0,0,1,0)
        | (x!!0) == ’F’ && (x!!1) == ’3’  = (0,0,0,0,0,1,0,0,0,1)
        | (x!!0) == ’G’ && (x!!1) == ’1’  = (0,0,0,0,0,0,1,1,0,0)
        | (x!!0) == ’G’ && (x!!1) == ’2’  = (0,0,0,0,0,0,1,0,1,0)
        |otherwise  = (0,0,0,0,0,0,1,0,0,1)



-- | The ’dupFilter’ function returns False if a given list contains the same
-- element more than once, returning True if the list is unique

dupFilter :: Eq a => [a] -> Bool
dupFilter [] = True
dupFilter (x:xs)
        | xs == [] = True
        | x `elem` xs = False
        | otherwise = dupFilter xs


-- | The ’correctFilter’ filters the given number of same pitches of the list

correctFilter :: Eq a => Int -> [a] -> [a] -> Bool
correctFilter n guess target
        | n == 0 = samePitchCount guess target == 0
        | otherwise = samePitchCount guess target == n


-- | The ’samePitchCount’ function returns the number of elements that are the
-- same, assuming there are NO duplicates

samePitchCount :: Eq a => [a] ->[a] -> Int
samePitchCount _ [] = 0
samePitchCount [] _ = 0
samePitchCount (x:xs)(y:ys) =
        if x == y
                then 1 + samePitchCount xs ys
                else samePitchCount (x:xs) ys + samePitchCount xs [y]
```

```
-- | The 'filterPitch' function deletes every same pitch which shown in the firs
t
-- list from the second list

filterPitch :: [String] -> [String] -> [String]
filterPitch xs [] = xs
filterPitch [] xs = xs
filterPitch (x:xs)(y:ys) =
        delete x (filterPitch (xs)(y:ys))


-- | The 'delete' function deletes the first occurrence of the element from the
-- list
delete :: (Eq a) => a -> [a] -> [a]
delete x [] = []
delete x (y:ys) =
        if x == y
                then ys
                else y : delete x ys


-- | The 'noteFilter' filters the given number of same note (with different
-- octave) in two lists.

noteFilter :: Int -> [String] -> [String] -> Bool
noteFilter n guess target
        | n == 0 = sameNotes == 0
        | otherwise = sameNotes == n
        where sameNotes = sharedNotes (filterPitch target guess) (filterPitch gu
ess target )


-- | The 'sharedNotes' returns how many same notes (with different octaves) two
-- lists have in common.

sharedNotes :: [String] -> [String] -> Int
sharedNotes _ [] = 0
sharedNotes [] _ = 0
sharedNotes (x:xs)(y:ys) =
  compareNote x (y:ys) + sharedNotes xs (delete (feedbackNote x (y:ys)) (y:ys))


-- | The 'compareNote' returns at most 1, which means the note find its pair in
the--  list.

compareNote :: String -> [String] -> Int
compareNote x [] = 0
compareNote x (y:ys)
   | compareNoteTuple (pitchTuple x) (pitchTuple y) == 1 = 1
   | otherwise = compareNote x ys


-- | The 'feedbackNote' function returns the note we find in the list, so in the

-- next turn we will delete it from the list.

feedbackNote :: String -> [String] -> String
feedbackNote x [] = ""
feedbackNote x (y:ys)
  | compareNoteTuple (pitchTuple x) (pitchTuple y) == 1 = y
  | otherwise = feedbackNote x ys



-- | The 'compareNoteTuple' returns how many notes in the two given tuples are s
ame--  but with different octaves

compareNoteTuple :: (Int, Int, Int, Int, Int, Int, Int, Int, Int, Int) -> (Int,
Int, Int, Int, Int, Int, Int, Int, Int, Int) -> Int
compareNoteTuple (a, b, c, d, e, f, g, one, two, three) (a1, b1, c1, d1, e1, f1,
 g1, one1, two1, three1)
        | (min a a1) + (min b b1) + (min c c1) + (min d d1) + (min e e1) + (min
f f1) + (min g g1) - (min one one1) - (min two two1) - (min three three1) == 0 =
 0
        | (min a a1) + (min b b1) + (min c c1) + (min d d1) + (min e e1) + (min
f f1) + (min g g1) - (min one one1) - (min two two1) - (min three three1) == -1
= 0
        | otherwise = 1


-- | The 'octaveFilter' filters the given number of same octaves (with different

-- note) of the list

octaveFilter :: Int -> [String] -> [String] -> Bool
octaveFilter n guess target
        | n == 0 = sameOctaves == 0
        | otherwise = sameOctaves == n
        where sameOctaves = sharedOctaves (filterPitch target guess) (filterPitc
h guess target )


-- | The 'sharedOctaves' function returns how many same notes (with different
-- octaves) two lists have in common, and the mechanism including the following
-- fucntions are similiar to 'sharedNotes'.

sharedOctaves :: [String] -> [String] -> Int
sharedOctaves _ [] = 0
sharedOctaves [] _ = 0
sharedOctaves (x:xs)(y:ys) =
  compareOctave x (y:ys) + sharedOctaves xs (delete (feedbackOctave x (y:ys)) (y
:ys))


compareOctave :: String -> [String] -> Int
compareOctave x [] = 0
compareOctave x (y:ys)
  | compareOctaveTuple (pitchTuple x) (pitchTuple y) == 1 = 1
  | otherwise = compareOctave x ys


feedbackOctave :: String -> [String] -> String
feedbackOctave x [] = ""
feedbackOctave x (y:ys)
  | compareOctaveTuple (pitchTuple x) (pitchTuple y) == 1 = y
  | otherwise = feedbackOctave x ys
```

```
-- | The 'compareOctaveTuple' changes a little of the 'compareNoteTuple' functio
n, -- since the formula will be minus 1.

compareOctaveTuple :: (Int, Int, Int, Int, Int, Int, Int, Int, Int, Int) -> (Int
, Int, Int, Int, Int, Int, Int, Int, Int, Int) -> Int
compareOctaveTuple (a, b, c, d, e, f, g, one, two, three) (a1, b1, c1, d1, e1, f
1, g1, one1, two1, three1)
        | (min a a1) + (min b b1) + (min c c1) + (min d d1) + (min e e1) + (min
f f1) + (min g g1) - (min one one1) - (min two two1) - (min three three1) == 0 =
 0
        | (min a a1) + (min b b1) + (min c c1) + (min d d1) + (min e e1) + (min
f f1) + (min g g1) - (min one one1) - (min two two1) - (min three three1) == -1
= 1
        | otherwise = 0
```