```
Beginning tests for mppan Wed Apr  3 22:52:53 AEDT 2019

Compiling sources
ghc -O2 -fno-omit-yields --make studenttest -o studenttest
[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 ...

Running tests
Haskell test run started 2019-04-03 22:52:55.464448483 AEDT
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
Available points: 20.0
Points earned: 20.0

Testing guessing functions
                    Guess test   1 ... PASSED 3.0
                    Guess test   2 ... PASSED 4.0
                    Guess test   3 ... PASSED 5.0
                    Guess test   4 ... PASSED 4.0
                    Guess test   5 ... PASSED 6.0
                    Guess test   6 ... PASSED 4.0
                    Guess test   7 ... PASSED 5.0
                    Guess test   8 ... PASSED 5.0
                    Guess test   9 ... PASSED 4.0
                    Guess test  10 ... PASSED 4.0
                    Guess test  11 ... PASSED 2.0
                    Guess test  12 ... PASSED 4.0
                    Guess test  13 ... PASSED 3.0
                    Guess test  14 ... PASSED 2.0
                    Guess test  15 ... PASSED 5.0
                    Guess test  16 ... PASSED 4.0
                    Guess test  17 ... PASSED 4.0
                    Guess test  18 ... PASSED 2.0
                    Guess test  19 ... PASSED 5.0
                    Guess test  20 ... PASSED 3.0
                    Guess test  21 ... PASSED 3.0
                    Guess test  22 ... PASSED 4.0
                    Guess test  23 ... PASSED 5.0
                    Guess test  24 ... PASSED 5.0
                    Guess test  25 ... PASSED 4.0
                    Guess test  26 ... PASSED 6.0
                    Guess test  27 ... PASSED 4.0
                    Guess test  28 ... PASSED 4.0
                    Guess test  29 ... PASSED 5.0
                    Guess test  30 ... PASSED 3.0
                    Guess test  31 ... PASSED 3.0
                    Guess test  32 ... PASSED 6.0
                    Guess test  33 ... PASSED 2.0
                    Guess test  34 ... PASSED 4.0
                    Guess test  35 ... PASSED 5.0
                    Guess test  36 ... PASSED 5.0
                    Guess test  37 ... PASSED 3.0
                    Guess test  38 ... PASSED 3.0
                    Guess test  39 ... PASSED 6.0
                    Guess test  40 ... PASSED 5.0
                    Guess test  41 ... PASSED 4.0
                    Guess test  42 ... PASSED 4.0
```

```
                    Guess test  43 ... PASSED 4.0
                    Guess test  44 ... PASSED 4.0
                    Guess test  45 ... PASSED 4.0
                    Guess test  46 ... PASSED 5.0
                    Guess test  47 ... PASSED 4.0
                    Guess test  48 ... PASSED 5.0
                    Guess test  49 ... PASSED 4.0
                    Guess test  50 ... PASSED 4.0
                    Guess test  51 ... PASSED 5.0
                    Guess test  52 ... PASSED 4.0
                    Guess test  53 ... PASSED 3.0
                    Guess test  54 ... PASSED 4.0
                    Guess test  55 ... PASSED 4.0
                    Guess test  56 ... PASSED 5.0
                    Guess test  57 ... PASSED 4.0
                    Guess test  58 ... PASSED 3.0
                    Guess test  59 ... PASSED 4.0
                    Guess test  60 ... PASSED 5.0
Total tests: 60.0
Tests successfully guessed: 60.0
Total guesses for successful tests: 246.0
Average guesses: 4.1
Available points: 50.0
Points earned: 50.0

Overall Results:
Available points: 70.0
Points earned: 70.0
Haskell test run ended 2019-04-03 22:52:57.047785111 AEDT
Total CPU time used = 1583 milliseconds
Completed tests Wed Apr  3 22:52:57 AEDT 2019
```

```haskell
-- File       : Proj1.hs
-- Author     : Ming Pan
-- Start Date : 28/03/2019
-- Purpose    : COMP90048 Declarative Programming Project 1 submission

-- Last Modified Date: 03/04/2019
module Proj1 (Pitch, toPitch, feedback,
    GameState, initialGuess, nextGuess) where
import Data.Char
import Data.List

-- A Pitch is composed by a Note (Char) and a Octave (Int)
-- The GameState coomposed by the different guesses (3-pitches chords)
-- The  first  guess  was concluded by numouros practice and it  can be
-- considered as the 'best' first guess
data Pitch     = Pitch { note :: Char, octave :: Int }deriving (Eq)
type GameState = [[Pitch]]

-- Define the consts
-- allPitches is the collection of all the valid pitches for gaming
-- allChords is the list of all the subsets of allPitches with length==3
-- firstGuess is conducted by numerous tests on the server and performs
--   the best (lowest average guess)
allPitches :: [Pitch]
allPitches = [Pitch note octave | note <- ['A'..'G'],
              octave <- [1..3]]
allChords  :: [[Pitch]]
allChords  = [chord | chord <- subsequences allPitches,
              length chord == 3]
firstGuess :: [Pitch]
firstGuess = [Pitch 'A' 1, Pitch 'B' 2, Pitch 'C' 2]


-- This is the function that takes String args and determine  whether it
-- is a valid input for generating a Pitch since a Pitch can be represe-
-- nted by 2-character String, the note of a Pitch ranged 'A' to 'G' and
-- octave ranged 1 to 3.
toPitch :: String -> Maybe Pitch
toPitch str
    | strlen == 2 && (note >= 'A' && note <= 'G')
         && (octave >= 1 && octave <= 3) = Just (Pitch note octave)
    | otherwise                          = Nothing
    where
        strlen = length str
        note   = head str
        octave = digitToInt (str!!1)

-- toString function that convert a pitch to a coresponded string
toString :: Pitch -> String
toString pitch = [note pitch] ++ [(intToDigit (octave pitch))]

-- Instance declaration for Pitch in the Show class
instance Show Pitch where
    show pitch = toString pitch

-- The feedback function takes the guess that provided by the  composer
-- and generate  the  result  of  correctness. The  rules  are shown as
-- follows:
-- 1. how many pitches in the guess are included in the target (correct
```

```
--     pitches).
-- 2. how many pitches have the right note but the wrong octave (correct
--     notes). Fomula: cn = |cp - sn|
-- 3. how many pitches have the right octave but the wrong note (correct
--     octaves) Fomula: co = |cp - so|
feedback :: [Pitch] -> [Pitch] -> (Int, Int, Int)
feedback target guess  = (nCorrPitches, nCorrNotes, nCorrOctaves)
    where nCorrPitches = length (intersect target guess)
          sameNotes    = findSame (eNotes target) (eNotes guess)
          sameOctaves  = findSame (eOctaves target) (eOctaves guess)
          nCorrNotes   = absMinus sameNotes nCorrPitches
          nCorrOctaves = absMinus sameOctaves nCorrPitches

-- eNotes is the  function that extract the  notes  of given pitches  and
-- combine them to a char list
eNotes :: [Pitch] -> [Char]
eNotes []     = []
eNotes (x:xs) = (note x) : eNotes xs

-- eOctaves is the  function that extract the notes of given pitches  and
-- combine them to a integer list
eOctaves :: [Pitch] -> [Int]
eOctaves []     = []
eOctaves (x:xs) = (octave x) : eOctaves xs

-- find the number of same elements of 2 given lists
findSame :: Eq a =>[a] ->  [a] -> Int
findSame [] e      = 0
findSame (x:xs) ys = if x `elem` ys
                     then 1 + findSame xs (delete x ys)
                     else
                        findSame xs ys

-- absolute result value from minus
absMinus :: Int -> Int -> Int
absMinus a b
     | a - b <= 0 = b - a
     | otherwise  = a - b

-- initialGuess fuction that take all the subsets for 3-pitch chords, and
-- remove the first guess from  the  game  state.
initialGuess :: ([Pitch], GameState)
initialGuess      = (guess, gameState)
    where
        guess     = firstGuess
        gameState = delete guess allChords

-- nextGuess function test all the candidates left in the game state with
-- the last guess made and save only the ones which is consistent to  the
-- correct answer. A possible target is inconsistent  with an answer  you
-- have received for a previous guess if the answer you would receive for
-- that guess and that (possible) target is different from the answer you
-- actually received  for  that  guess.  Once  the   game  state has been
-- updated, use the information to get the best option for next guess
nextGuess :: ([Pitch],GameState) -> (Int,Int,Int) -> ([Pitch],GameState)
nextGuess (lastGuess, gameState) score = (nextGuess, newState)
    where
        sameScoreCands = [target | target <- gameState,
                          feedback target lastGuess == score]
```

```
        newState       = delete lastGuess sameScoreCands
        nextGuess      = bestGuess newState

-- The bestGuess function return the best guess (the possible target that
-- will leave the  maximum  number  of  possible  targets if we guess it)
-- which is the first element (with the greatest number) of  the  sorted
-- tuple list [(possTarget, numOfTarLeft)].
bestGuess :: GameState -> [Pitch]
bestGuess gameState     = fst (head sortedTup)
    where
        tupNumOfTarLeft = getTup gameState gameState
        sortedTup       = sortBy sortRules tupNumOfTarLeft

-- This is the function that customized the ordering rules. The  function
-- compares  the  second element of a tuple and decide the order. In this
-- case, it is the possible target left. The greater the  number is,  the
-- more chance that target is the actual target.
-- This can be explained as:
--   totalTarget = scoreCase1 * m(1) + scoreCase2 * m(2) + ..
--               + scoreCasen * m(n)
--   1. totalTarget is the total possible target in a game state (const),
--   2. scoreCase1 to scoreCasen are different score types  from  testing
--   3. the m(1) to m(n) are the numbers those relate to their score type
--      respectively.
--   let's suppose scoreCase m is the correct answer.Then we can conclude
--   that the more scoreCases, the smaller m (x) from probability study.
sortRules :: (Ord b) => (a, b) -> (a, b) -> Ordering
sortRules x y
    | (snd x) < (snd y)   = GT
    | otherwise           = LT

-- The getTup returns a tuple  composed by  2 elements. The first element
-- represents the possible target and the second  part shows  the   total
-- number of score types leave if we guess with it.
-- The process is described as follows:
-- 1. Get the possible target (possTar).
-- 2. Test each other chord (in the game state) with that possible target
--     (possTar) and store the score into a score list.
-- 3. Sort the score list so that we can group the same result.
-- 4. The length of the groupted list is the number of total different
--    score types of which corresponding to its' target  (possTar)
getTup :: [[Pitch]] ->  GameState -> [([Pitch], Int)]
getTup [] _             = []
getTup (x:xs) gameState = (x, scoreTypes) : getTup xs gameState
    where
        newState        = delete x gameState
        scoreTypes      = length (group (sort [score | guess <- newState,
                          let score = feedback x guess]))
```