-- %%%%%%%%%%%%%%%%%%%%%%%   Part A   %%%%%%%%%%%%%%%%%%%%%%%

makeHelper4 :: Char -> [Char] -> [Char]

makeHelper4 c (x:xs)

Takes a Char c and a List of Chars (x:xs) (String) and returns a list of Chars (String) for any Char that comes just after the Char c

makeHelper5 :: Char -> [[Char]] -> [Char]

makeHelper5 c (x:xs)

Takes a Char c and a List (x) of list of Chars (x:xs) (Calls makeHelper4) returns a list of Chars (String) for any Char that comes just after the Char c in all the list (x:xs)

makeHelper6 :: Char -> [Char]

makeHelper6 c

Just calls makeHelper5 for the Char c and list training from music resources

pack ::Char -> [Char] -> [(Int,Char)]

pack c xs

Takes a Char c and return a pack contains the number of repetition of the Char c in the list xs in a pack (Number of repetition, Char c)

possiblePack1 :: [Char] -> [Char] -> [(Int,Char)]

possiblePack1 (x:xs) cs

Find all the possible packs for list chars in training in the list cs given form makeHelper6

possiblePack :: Char -> (Char,[(Int,Char)])

possiblePack c

calls the function possiblePack for all chars and sorts it in a descending order

makeStatsList :: [(Char,[(Int,Char)])]

makeStatsList

Maps the function possiblePack for all chars

-- %%%%%%%%%%%%%%%%%%%%%%%%   Part B   %%%%%%%% %%%%%%%%%%%%%

sumf :: [(Int,Char)] -> Int

sumf (x:xs)

Gets the sum of the first element in the in the pair(x) and does it for list (x:xs)

findELm :: Int -> [(Int,Char)] -> (Int,Char)

findELm c (x:xs)

takes a random number as c and finds the position that the cumulative sum of elements in the list sums up to this number

getElm1 :: [(Int,Char)] -> (Int,Char)

getElm1 x

Takes a list x calls findElm element with c equals to a random number from 1 to sum of the list x

getElm :: [(Int,Char)] -> Char

getElm x

get the second element of the pair

findx :: Char -> Char

findx c

calls findx1 of c and a list generated by makeStatsList

findx1 :: Char -> [(Char,[(Int,Char)])] -> Char

findx1 c (x:xs)

Takes a Char c and searches in the list (x:xs) for the element that contains c as it first parameter then calls getElm.

Handels the Case if a number is chosen that have no numbers after it, therefore it throws an error “Dead End”

compose :: Char -> Int -> [Char]

compose x k

Append x with the result from compose1

compose1 :: Char -> Int -> [Char]

compose1 x k

calls findx for Char x to find a random char and appends it with other random Chars until the length of list (String) is K

remove :: [(Char,[(Int,Char)])]

remove

Calls remove1 for list makeStatsList

remove1 :: [(Char,[(Int,Char)])] -> [(Char,[(Int,Char)])]

remove1 (x:xs)

Checks if the second element of pair is empty [] it removes it form the list

contains :: Char -> Bool

contains c

calls contains1 for Char c and list from remove

contains1 :: Eq a => a -> [(a,b)] -> Bool

contains1 c (x:xs)

checks if the Char c is an element of the first pair in the list of list of pairs (x:xs)