import Data.List  
import MusicResources  
x=chars  
y=chars  
n= (length chars) -1

---------------------------------Part A--------------------------------

getModule element 0 =[element++[y!!n]]  
getModule element count =(element++[y!!count]):getModule element (count-1)

getModule1 element= getModule element n

findMinY [] \_ n=n  
findMinY \_ [] n=n  
findMinY \_ (l:[]) n=n  
findMinY m (l:(r:t)) n = if (m!!0==l) && (m!!1==r) then findMinY m(r:t) n+1 else findMinY m(r:t) n

freq\_count m = findMinY m (concat training) 0

brackets list 0 =[(freq\_count(list!!n),((list!!n)!!1))]  
brackets list count = (freq\_count(list!!count),((list!!count)!!1)):brackets list (count-1)

brackets1 list = brackets list n

removeZeros::[(Int,Char)]->[(Int,Char)]  
removeZeros []=[]  
removeZeros (h:t) =if(fst(h)==0) then removeZeros t else h:removeZeros t

desc\_sortedBrackets count g = reverse (sort (removeZeros (brackets1( g))))

makeStatsList1 0 = [( x!!0, desc\_sortedBrackets 0 (getModule1 [x!!0]) )]  
makeStatsList1 count = ( x!!count, desc\_sortedBrackets count (getModule1 [x!!count])):makeStatsList1(count-1)

makeStatsList::[(Char,[(Int,Char)])]  
makeStatsList= reverse(makeStatsList1 n)

**T16G07 Code**

---------------------------------Part B--------------------------------

decompose (0,k)= []  
decompose (h,k)= k:decompose(h-1,k)

getString []=[]  
getString (h:t)= decompose h:getString t

getString1 w = concat (getString w)

get\_list c count m = if(c==fst(entry)) then snd(entry) else get\_list c (count+1) m  
 where entry = m!!count

get\_list1 c m =get\_list c 0 m

get\_char0 s=if s == [] then error "Stuck .. no possible next character" else s!!randomZeroToX(length s-1)

get\_char c m = get\_char0 (getString1(get\_list1 c m))

compose::Char-> Int->[Char]  
compose char 1 = [get\_char char makeStatsList]  
compose char i = char : compose(get\_char char makeStatsList) (i-1)

**T16G07 report**

x=chars  
y=chars  
n= (length chars) -1

|  |  |  |
| --- | --- | --- |
| Function | Description | Inputs |
| getModule element count | Returns a list of modules for a certain element  [“00”,”01”..”0z”] | element: an element from chars string  count: a counter |
| getModule1 element | Calls the function getModule but starts the counter from ‘n’ (length of chars string imported from MusicResources) | element: an element from chars string |
| findMinY m ( l : ( r : t) ) | Finds the frequency of a certain module in a list | m: module obtained from getModule1 ex.”0e”  l: head of the list  r: head of the tail list  l: tail of the tail list |
| freq\_count m | Calls the function findMinY on the concatenated list of training imported from MusicResources | m: module obtained from getModule1 ex.”0e” |
| brackets list count | Creates a list of the statistics pairs | list: list of modules for a certain element ex. [“00”,”01”..”0z”]  count: counter |
| brackets1 list | Calls the function brackets but starts the counter from ‘n’ | list: list of modules for a certain element ex. [“00”,”01”..”0z”] |
| removeZeros ( h:t ) | Removes the pairs with 0 frequencies from the list generated by brackets1 function | h: head of list  t: tail of list |
| Desc\_sortedBrackets count g | Sorts a list descending | count: counter  g: list of modules |
| makeStatsList1 count | generates a list of statistics according to the content of the training list in the MusicResources.hs ﬁle. | count: counter |
| makeStatsList | Calls makeStatsList1 but returns an ordered output and starts the counter from n |  |
| Decompose (h,k) | This function takes a pair and returns a list of all the possible occurrences of this character | h: is a number of occurrence  k: is the char that occurred h times |
| getString (h:t) | The function gets the list from Decompose and produce a list of strings separated by commas of the char equivalent to the number of occurrences | h: head of the list  t: tail of the list |
| getString1 w | it concats the list we produces to give us a string | w: string without any separation |
| get\_list c count m | This function search if we found the character we want else we increment the counter and it search in the next position | c: the character we’re searching for  count: to keep track of our position  m: this is the makeStatsList |
| get\_list1 c m | This is the base function for get\_list | c: character we’re searching for  m: makeStatsList |
| get\_char0 s | This function gives error if there are no characters after it else gives ta random character found in the list | s:string |
| get\_char c m | This function produce a random character from the string we go from the list we produced | c: character we’re looking for  m: makeStatsList |
| compose char i | This function produce a string of possible outcomes with the number we have given the program | char: character we want  i: number of characters we want to compose |