**Music Composition Report**

**A Haskell implemented projected done by:**

**Ahmed Anwar 31-3487**

**Moustafa Nagaty 31-2075**

**Youssef Zidan 31- 3719**

The Project was split into two parts, the first part was to gather and learn statistics on the sample music pieces provided, the second part was to use these statistics to compose music on a semi random selection basis using the data in part one.

**Part A**

The function makeStatsList is the main function responsible for making these statistics. It depended on a number of low level functions:

1. flatten: it is used to ‘flatten’ a list of lists into one list.
2. flatTrain: it uses flatten to flatten the ‘training’ list.
3. Occurrences: it checks the number of occurrences of 2 characters concatenated to each other.
4. occMaker: It takes a character and a list as arguments, it should expect the list to be ‘chars’, first it implicitly concatenates the input character with every element in the input list then it calls ‘occurrences’ on this concatenation and flatTrain, if it doesn’t find any occurrences it completes its recursion without adding anything to the output list, if it does, then it constructs a tuple of the number of occurrences of this concatenation and the second character in the concatenation, doing so for the rest of the list.
5. Insert1: It takes an tuple and a list of tuples and and checks the first element in both tuples (which is the frequency) and then sorts them in descending order.
6. Sort2: it takes a list and sorts it descending.

**Part B**

The function compose is the main function for composing the music by using the output of part ‘A’ and making use of it with the addition of using other sub functions that is related to the composition of the output piece.

1. composeHelperSum: It takes a list of tuples and returns the sum of every first element of every tuple.
2. ComposeGetLetter: It takes a list of tuples and two integers, if the first input integer is bigger than the first element in the current tuple, it subtracts the second input integer from the first element in the current tuple and calls itself with this result and the tail of the list, if any of the calls had met any other case then it returns the second element of the current tuple.
3. getList: Takes a char and a list of tuples and returns the second element in the current tuple if the input char is equal to the first element of the current tuple.