Computer Science 229
Homework Project 1 – Part 2
1600 Points

Note

This part of the assignment is dependent on the first part in the sense that it requires you to write code that reads and writes sound files in various formats. Many of the parts of this assignment are relatively easy provided that you have already written functions that perform this reading and writing. Nevertheless, the points allocated to these parts are relatively high so that if you did not receive full credit on Part 1, it is still valuable to finish/correct that part in order to receive credit for this part. And, in order to further motivate you and give you the chance to receive some credit on this first critical part, you will be allowed to turn in Part 1 again to receive additional credit. NOTE: if you decide to do this, your ENTIRE Part 1 will be graded again (likely with additional or different test cases) and your score for Part 1 will be the AVERAGE of the two submissions. This means that your score could be LOWER.

The Assignment

Part E) (200 Points) Write a program named "mix" that inputs 2 sources of sound data, computes the average signal for each channel (mixes the two sources) and outputs the result in a specified format. More precisely, if s1 and s2 are the values of samples from source 1 and source 2, respectively, then the "mix" of s1 and s2 is (s1 + s2)/2; i.e., the average of s1 and s2. Then syntax of the mix command when it is

```
mix –t fmt ss1 ss2
```

where ss1 and ss2 are the names of sound files in either format. (The mix program must determine the format type on its own.) The format fmt is either "wav" or "cs229". Note that if the "-t" parameter is not present then the output is assumed to be in the "wav" format. In any case, the output in the required format is sent to standard output.

Note that for multiple channel sources (such as stereo) the mix is performed separately to each channel.

Part F) (100 points) Write a program named "rms" that given a sound file (of either format) read from standard in, outputs a single number that is the RMS (root mean squared) value of all the samples. This command only works on mono (single channel) sound files. An error must be generated to stderr in this case, and the program should return an error code indicating that there is an error. The program must also return a success code of 0 along with the correct output if the program succeeds. The RMS value of n numbers is defined as:

$$\sqrt{\frac{\sum_{i=1}^{n}(S_i)^2}{n}}$$

That is, the square root of the sum of all the samples squared divided by n.

Part G) (100 points) Write a program called "scale" that reads a sound file (of either format) from standard input, outputs a sound file in the same format that is scaled by a value given on the command line. The format of the command is

```
scale val
```

where val may be any positive floating point number. If the scaling results in a sample value that is larger than the maximum number supported by the sound file format, then the sample is clipped to the maximum value supported by the sound file format.

Part H) (200 points) Write a bash script named "normalize" that given a sound file name as a parameter, outputs a new sound file in the same format with an RMS value of one half the maximum range (max – min) offered by the sound file format. Note that you must use the programs "rms" and "scale" appropriately. You may if you wish, write and use one additional C program that given a sound file as input, outputs the maximum range given the format.

Part I) (200 points) Write a bash script named "dial" that utilizes gensine and mix to output the DTMF tones given a telephone number as input. Example:

```
dial 5152945817
```

will output a single sound file that would actually dial the telephone number if played through a landline phone. Hint: you will likely want to create a C program (maybe called scat) that combines (concatenates) two or more sound files of the same format into a single sound file. You may also want to utilize temporary files for this, but be sure to clean them up before your script terminates.

Part J) (200 points)

Write a Java program named "dft" that reads a file of integer numbers, up to 4096 numbers maximum, and then outputs the DFT of the sequence of numbers at a specified frequency. For example:

```
dft xyzzy 440
```

reads the samples contained in the file xyzzy and then outputs the DFT of the sequence of samples at the frequency of 440 hz.

IMPORTANT: To receive any credit for this part, the Java program must call a C program via JNI (Java Native Interface) to compute the DFT.

Part K) (500 points) For this problem you will be writing a Java program to tune a piano. There are several requirements for this part, including a GUI interface for the user, JNI for DFT computation, and input of sound samples. The GUI interface consists of several components or controls. The first control allows the user to select which of the 88 keys on the piano to tune. For a list of these frequencies and key numbers see

http://en.wikipedia.org/wiki/Piano_key_frequencies.

The sue must be able to easily select an input file to process. Finally, the results of the processing displays how in-tune the input file samples are compared to the selected frequency (piano key number). This display is a needle that when centered means that the samples read from the file are exactly in tune. If the needle is to the left of 0 then the samples are flat (in-tune with a frequency lower than the desired frequency), while right of 0 the sample are sharp (in-tune with a frequency that is higher than the desired frequency).

To determine the frequency the samples represent, you must take the DFT of the samples around the center frequency at ½ frequency increments. The range is -20% to +20% of the center frequency. To determine the dominate frequency given the set of DFTs defined above, find the value with the maximum energy. However, you must also determine if there is a peak over all frequencies and display an indication that the sound is so far from having a tone at that frequency, it cannot be displayed. There are several ways to do this, including defining a peak to be some factor above the average.

You may choose or not to choose to use JNI to read in sound files. Presumably you could easily adapt your existing C functions for reading sound samples into a Java array.

You may choose or not to choose to use JNI to compute the DFT at specific frequencies, but again, if you already have part J completed, this should be easy.

Part L) (100 points) There must be a complete makefile turned in.

Documentation/Style is worth at least 15% of your grade. Note that copying and pasting the same code through several functions will likely lose most of these points.