Computer Science 229
Homework Project 1
2100 points

C Programming Project. (Fun with digital sound)

Notes

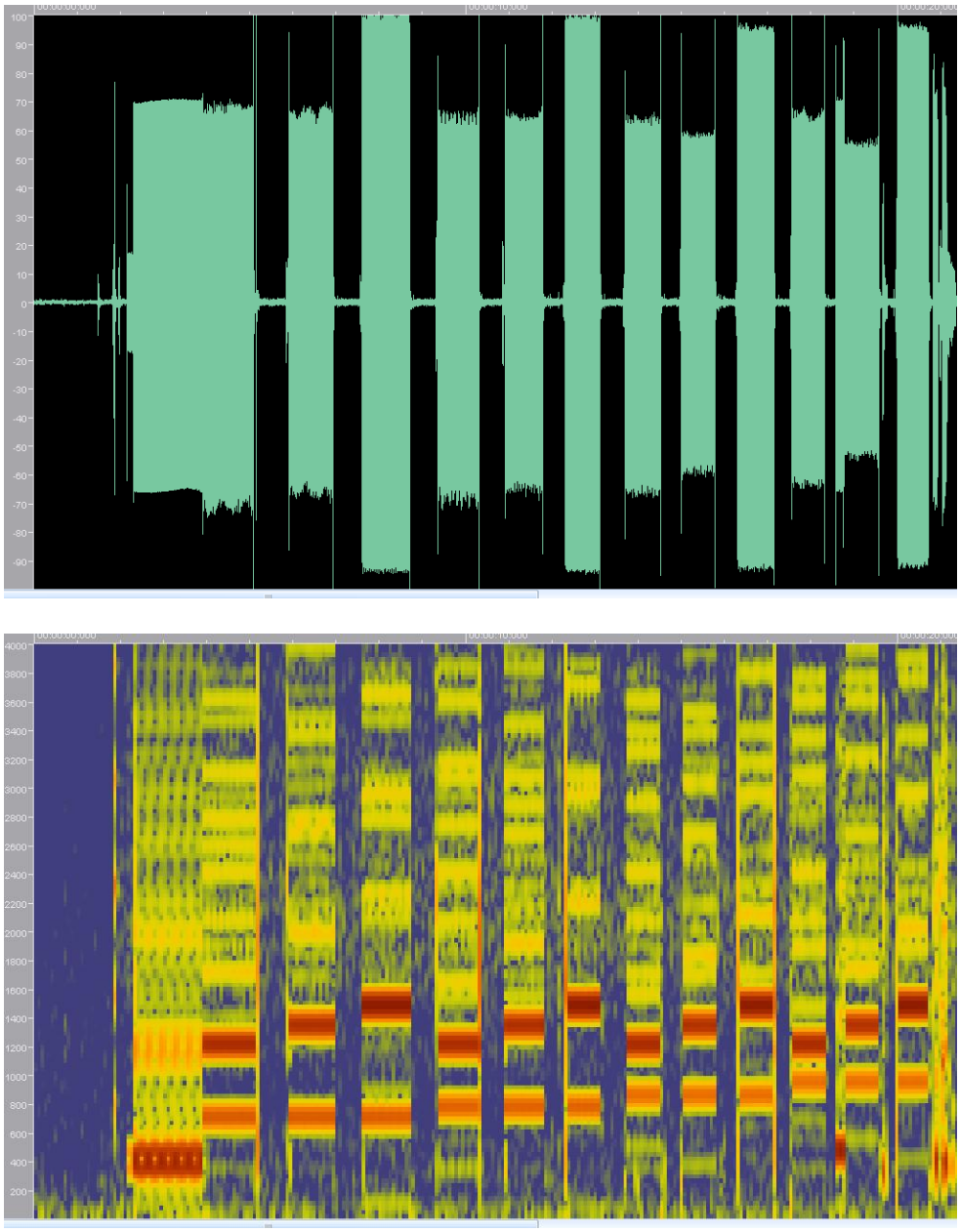
Aspects of this project contain a little math. This is not a class in signal processing nor the mathematics of signal processing. However, in an attempt to create somewhat interesting assignments for both engineers and computer scientists, the programming projects we will do in this class are either real-world applications, or approximations of them. However, I do not expect you to derive and understand the math in some of these projects. As you read through this assignment, there are several places where certain aspects of the program will be discussed in class. This means that I will probably give you big hints or tell you exactly how to do it. You should take this as a strong hint to try not and miss class. If you do miss class, make sure you have a friend or someone in class that will share their notes with you. (I am certain that my notes will not be legible to even super human beings.)

Introduction

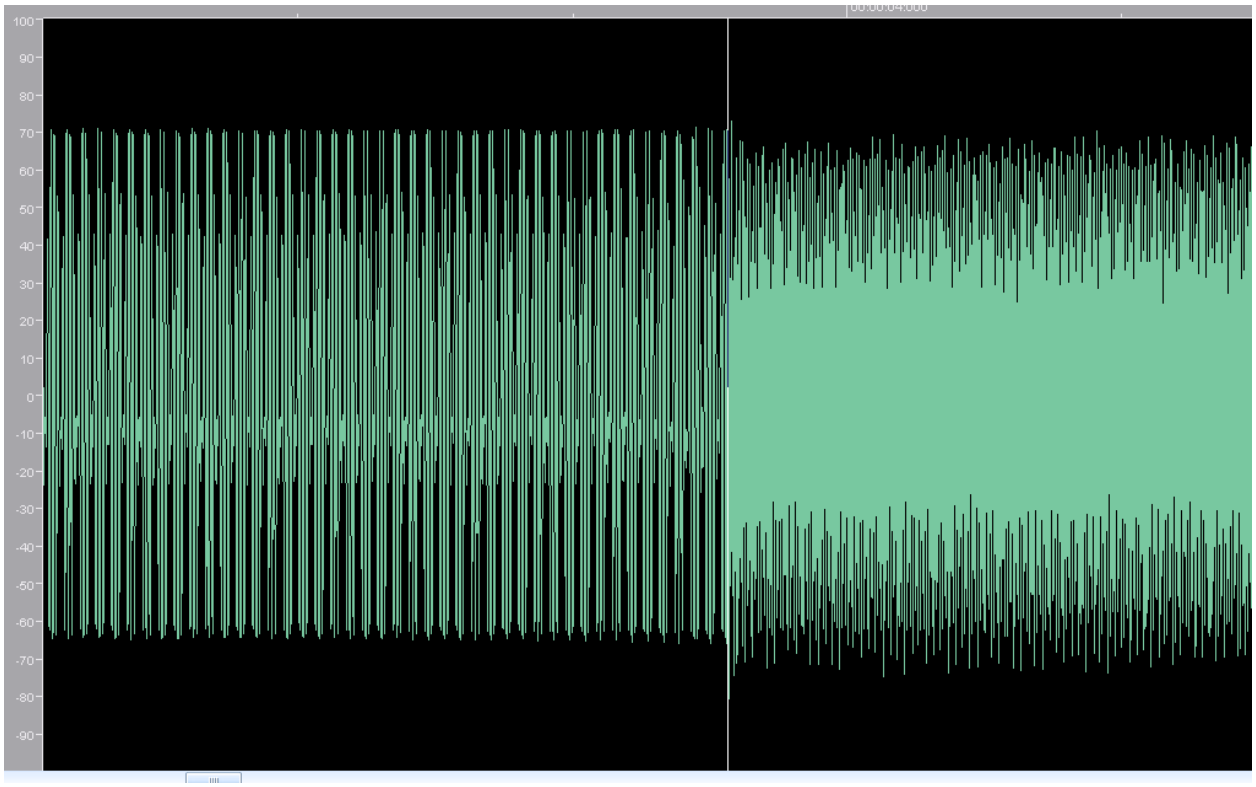
This project utilizes digital sound processing techniques to perform various tasks on small sound samples. Again, you are not responsible for understanding the mathematics behind formulas and algorithms given to you, but you are required to translate these formulas and algorithms into functions and working programs.

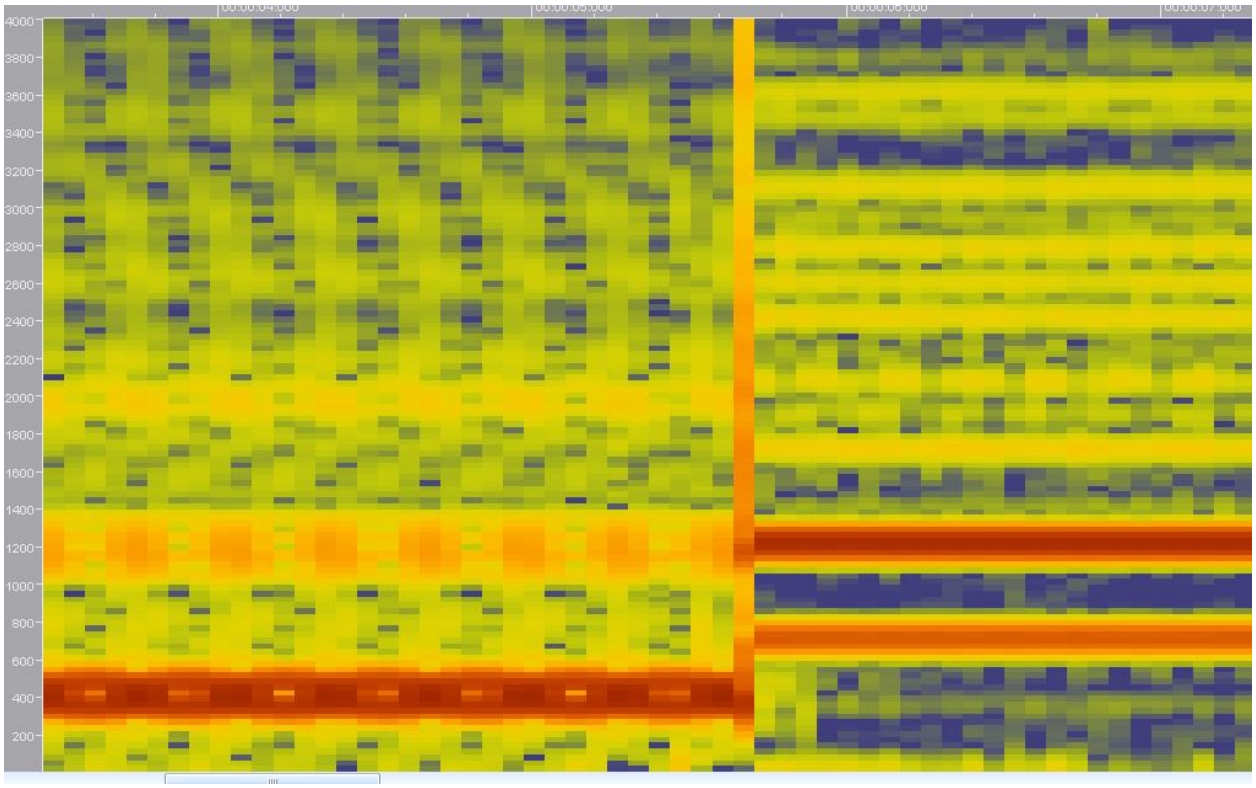
Sound Background Information

Sound is simply a wave of changing air pressure. This change in air pressure is detected inside the ear and translated into what we hear. For a simple sine wave of a frequency of 440 Hz, you would hear a very pure single tone. However, most sounds are complex and do not follow any predictable pattern. The natural way to view sound waves is by looking at the amplitude versus time. However, another way to view sound is how much "energy" is in each frequency. The two pictures below show the wave amplitude and the corresponding frequency energy of a sampled sound signal of DTMF tones. (DTMF are the tones used in phone systems to transmit the key pressed.) In the frequency energy graph, the more red the color indicates the more energy at that frequency.



A magnified image is shown below for a portion of the dial tone (440 Hz – go figure) and the tone when the number 1 button is pressed on a touchtone phone.


100
90
80
70
60
50
40
30
20
10
0
-10
-20
-30
-40
-50
-60
-70
-80
-90


4000
3800
3600
3400
3200
3000
2800
2600
2400
2200
2000
1800
1600
1400
1200
1000
800
600
400
200

A digital recording uses analog to digital converters to translate the electrical signals that correspond to air pressure into binary numbers that can be read easily by computers and signal processors. Note that if you have digital sound data, you can easily make the entire sound louder or softer by multiplying all the numbers by a constant number. (A number greater than 1 will make it louder, and a value less than one will make it softer. Can you explain why adding a constant value to all the numbers does not change the sound that you hear?) Sound data is often stored on computers in files in a variety of formats, including MP3, WAV, etc. Since it can be very complex to figure out how to use external libraries in C, you will not be required to do this for this project. (You may be required to do this in homework 2 which will use classes and C++.) In this project, you will simply read integer numbers from a file.

Also understand that each digital sample of sound is taken at a particular time. For a sequence of samples as is normally the case, it is important that each sample be taken at equal time intervals. This is called the sample rate and is given in terms of samples per second. (Often called a Hertz) For example, the samples above were taken at 8000 samples per second, or 8000 Hz, or 8 KHz. (Kilohertz). A well-known theorem says that the highest frequency you can reproduce with a digital sample is half of the sample rate. Thus, the highest frequency that can be recorded with an 8 Kilohertz sampling rate is 4 Kilohertz. Compact disks that store music have digital data that is stored at 44.1 Kilohertz and thus the highest frequency sound that a CD can play is 22 Kilohertz. (Most human can only hear up to about 20 Kilohertz.)

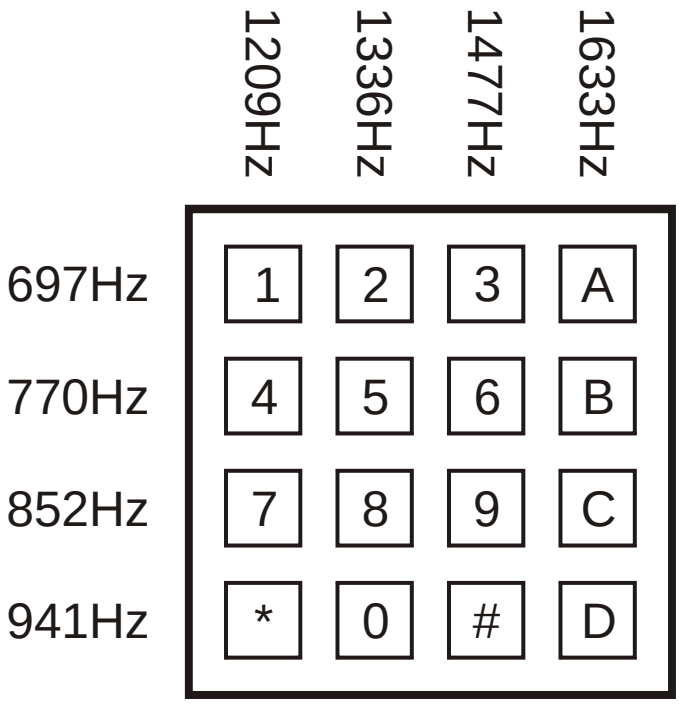
DTMF and your phone

DTMF stands for Dual-Tone Multi-Frequency and is most commonly used in telephone dialing and signaling. In this homework, you will write C programs to process sound samples and perform various tasks, including encoding and decoding DTMF signals.

We begin with an explanation of DTMF. Using 8 different tones (sound frequencies), we can encode 16 different code-words (characters, keys, etc) by utilizing a 4 by 4 array and labeling the rows and columns with the frequencies. Using the following frequencies,

697 Hertz,
770 Hertz,
852 Hertz,
941 Hertz,
1209 Hertz,
1336 Hertz,
1477 Hertz,
1633 Hertz,

we can create the following array:



It is not surprising that part of this array looks very much like a telephone keypad. In fact, when you press the "1" key on a touch-tone telephone pad, the telephone generates a tones composed of a 697 Hz tone and a 1209 Hz tone. This tons is received by the telephone equipment at the other end of the telephone line and is used to determine the number you are dialing, or in more recent years, to navigate phone trees.

For this type of signaling, our project will be concerned with a few other aspects of this tone. The actual specification for these tones is much more involved and requires much thought, engineering, and mathematics in order to decode according to that specification. This is a very simplified, but functional version. Note that your program may not even meet these simplified specifications. It is part of the assignment to experiment and find a reliable program for decoding these signals.

1. How long is the duration of the button-press (tone generated) before the equipment (your program) recognize that the tone is present and the button has been depressed?

2. How strong (loud) must the tone be to be recognized? Is this loudness absolute, or is it with respect to the noise in the signal?

3. After a tone has been detected, how long is it required to be absent before the presence of the same tone constitutes a second button push of the same button?

Some DTMF Specifications

Signal duration operation (Minimum time required that a tone signal is present)

40 milliseconds

Signal duration non operational (Time that the tone is not received when receiving a tone before the tone can be considered not present)

23 milliseconds

Signal interruption (Maximum time a tone can be interrupted and still must be considered a single tone.)

10 milliseconds

Notice that the time between 10 and 23 milliseconds is not well defined. In this case a gap of this time may be considered either 1 or 2 tone detections.

Question: At a sample rate of 8 KHz, how many audio samples are there in 10 milliseconds?

Finally, there is a specification for the actual frequencies and how high above the noise level a the energy level must be for the tone to be detected. DTMF tone frequency is actually a range of frequencies that is within 1.5 percent of the center frequency. . That is, for a signal of frequency of 697 Hz, a tone would be detected for any tone frequency from 697 – 0.015 * 697 to 697 + 0.015 * 697. Similarly, there should not be any energy that is more than 3.5 percent from the center frequency. This is how we determine if a signal is above the noise level. A tone is present if there is at least 5 times the energy at the center frequency than at the two frequencies that are 3.5% away from the center frequency.

Important Note: DTMF tone detection is an optional part of this assignment, and it is not easy. I have included it here because students have completed it in the past, learned a lot, and had fun doing so. It is not clear that the simple methods described here to detect these signals will work even for these simplified requirements. It is not even clear that the real algorithms used can meet these requirements, but we will give it a try. We may as a class decide to relax some of these requirements (especially the frequency requirements) in order for this to work. Do come to class and check WebCT for changes in the requirements. Your code should be written in a way such that if a change is required, it should takes less than a few minutes to make the change in your code. This is one of the marks of a well-written program -- The programming anticipates changes in requirements like this. This assignment is written this way on purpose in order to help you learn to think ahead in programming. In other words: Do not get mad if suddenly we decide to use 32 KHz samples instead of 8 KHz samples. It is promised that such changes will not occur within 24 hours of the due date, provided there are significant numbers of students working on the project before that.

Detecting the Energy at a Frequency

There are many ways to measure the amount of energy at a particular frequency, some more accurate than others. If you search the Internet for DTMF algorithms you will no doubt find lots of code for solving this problem and even this particular assignment. **Use of any of these algorithms or programs without prior consent of the instructor or not providing proper credit to the author in the assignment, will get you a student conduct charge for academic dishonesty and an F in this class**. However, I will give you a method to determine an indicator for the energy at any particular frequency. Note that you would never use this method in a real-time system such as a phone exchange as it requires a lot of processing time, but it is simple and easy to implement.

The energy estimate E is

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{i 2 \pi n k / N}$$

where,

x is the sample data
N is the window size,
k is the kth frequency component, and
i is the square root of -1.

This equation will be discussed in detail in lecture.

E(k) is simply the magnitude of X(k) (X(k) is a complex number)

Because the math is quite involved here, the simple formula you are looking for is that the frequency is given by k (k < N/2) in steps of the sample frequency divided by N. If you had 100 samples from time 0 to time 99 and chose a window size of 10 using an 8 KHz sample frequency. Then,

E(0) is the energy at frequency 0.
E(1) is the energy at frequency 800
E(2) is the energy at frequency 1600
E(3) is the energy at frequency 2400
E(4) is the energy at frequency 3200
E(5) is the energy at frequency 4000

Note that E(6) through E(9) are not used and do NOT correspond to frequencies above 4000.

You will need to choose a window size large enough to achieve the desired resolution for each frequency, but small enough as to not introduce long latencies. (also discussed in class)

Sound File Format

In this assignment you will be required to read and write files that contain sound samples. In addition to the raw data, the file must also contain a header portion that specifies various parameters about the samples in the file.

The header portion is positioned at the beginning of the file before any sound samples. A header consists of the keyword header in upper or lower case, or mixed. All characters before the header are ignored. The header keyword resides on a single line.

Once the header keyword is found, subsequent lines of text specify the parameters of the sound samples. (Special note: the word sample refers to the number of pieces of data at a specific time. Therefore a stereo sound sample consists of two numbers while a mono sound sample consists only one numeric value.) These parameters are of the form keyword followed by whitespace (tab or space characters) followed by the parameter value. An example of the file format below will make this clear. After all header information is specified, the header section must end with a single line with the word endheader in upper or lowercase.

Header information may come in any order, and some header keywords have default values if they are not present while others must be present. If a keyword is not present when required, or if there are unknown keywords, or if the parameters are invalid in any way, then the file format is in error. Keywords and parameters are placed one per line. Valid parameters are:

FREQUENCY <sample frequency> required.
SAMPLE <number of samples in file> optional
CHANNELS <STEREO | MONO> optional
BITRES <number of bits in a sound value> required

If the SAMPLE field is specified the file must contain that number of samples or the file is in format error. If it is not specified, then samples are read until the end of file is found. Note that for stereo sound samples, this field represents the number of stereo pair samples to read. For example, if the SAMPLE field indicates 10 samples and the CHANNELS filed indicates stereo, then 20 sound values are read.

If the CHANNELS field is not specified then the default is MONO.

The BITRES field is required but is limited to the following values: 8, 16, and 32 bits per sample.

After the header information, data values are represented in binary format from least significant byte to most significant byte. Thus for 8-bit mono samples, a single byte (char) value is written for each sample. For 16 bit mono samples, two characters are written for each sample, the first character represents bits 0 through 7 and the second byte represents bits 8 through 15. Samples are always positive. If the sound samples represent a stereo signal with both a right and left channel, then both samples are placed in the file with the left channel before the right channel.

The Assignment

For all parts of this assignment, be sure to separate functionality into separate files and make sure your file structure is documented in a README file. For example, you will likely need to read and write data in the file format above in several different places. It makes sense to place the reading, writing, decoding, and encoding of the sound files into a separate file so that the functions may be easily used in other places.

Part A) (200 points) Write 4 different C programs called gentest1, gentest2, gentest3, and gentest4 that outputs test sound file with the following characteristics.

gentest1 outputs a sound file in the correct format in 8-bit resolution mono with 10,000 samples at a frequency of 8000 samples per second. The sound samples are all zero.

gentest2 outputs a sound file in the correct format in 8-bit resolution stereo with 2,000 samples at a frequency of 16,000 samples per second. The left samples are all zero value and the right samples are all a value of 200.

gentest3 outputs a sound file in the correct format in 32-bit resolution mono with 40,000 samples at a frequency of 8,000 samples per second. The samples all have a value of 20.

gentest4 outputs a sound file in the correct format in 16-bit resolution stereo with 20,000 samples at a frequency of 44,100 samples per second. The left samples all have a value of 1000 and the right samples are all a value of 4242.

Include a simple makefile with a target of “tests” that will build all four of these test files.

Part B) (200 points) Write a C program called “info” that checks if a sound file is in the proper format and then outputs to standard output the name of the file, the sample frequency, whether the file represents stereo or mono sound, the number of bits per sample, and the number of samples in the file. If the file is not in a valid format, then an error message is output that indicates the first format error found in the file. Note that the sound file is input through stdin, and the easiest way to test this part is to use I/O redirection of a valid file.

Modify the makefile from part A to include a default target of “all” that makes all the programs from parts A and B. The target for part B must be named “info” the same as the program name.

Note that it is also a file format error if a sound sample value is not in the proper range for the bitsize.

Part C) (200 points) Write a program called gensine that outputs a sine wave of the specified frequency, amplitude, sample rate, bit resolution, and duration to standard output in the format described above. The syntax for the command line is as follows:

gensine <MONO | STEREO> <frequency> <bit-size> <sample rate> <amplitude> <duration>

Where the frequency is specified in hertz, bit-size in bits (8, 16, or 32 bits) sample rate in samples per second, and duration is specified in seconds and may be a floating point number.

Remember that you can use output redirection to store the output generated by your program in a file so that you can more easily verify that the output is correct.

Update the make file so that there is a target named "gensine" that creates this executable file. Also update the makefile so that "make all" generates all the executable files from parts A, B and C.

Part D) (100 points) Write a bash shell script called "sinfo" that has a parameter that names a file name and then outputs the "info" for that file. For example, the user may type "sinfo sound.dat" and the output should be the same as if the data in sound.dat were given to the info program as input. Note that you can write this script file in less than 3 lines.

Part E) (100 points) Write two programs called "split" and "combine". Split reads a mono stream in the format above from standard input and outputs the samples in stereo format to standard output by duplicating the mono channel into both stereo channels. The combine program takes a stereo input stream from standard input and outputs a mono stream by combining the two stereo channels into one by averaging the left and right channels (left + right)/2 to form the mono output. Your programs must check for valid input file format and report appropriate errors to stderr. Modify your makefile appropriately so that "make all" produces all the executable files from parts A through E. The two make targets for the executable files for this part are "split" and "combine".

Part F) (50 points) This part is similar to the gensine program except that you are required to output a random value of the appropriate size for each sound sample. The program is called static and has two command line parameters. The first is the number of bits for each sample (8, 16 or 32) and the second is the number of samples to create. The output of this program always generates a MONO type sound file. As usual modify the makefile appropriately.

Part G) (200 points) Write a C program called mix that takes a list of files in the above format and a list of relative gains on the command line and performs a mixing function on all the file data. The output is sent to the standard console output in the same format as described above. An example command line of this format is shown below.

mix sound1 2.0 sound2 1.5 sound3 3.0 sound4 4.0 > mixsound

All input files must be of the same format and of the same type MONO or STEREO. An error message is displayed if a MONO file and a STEREO file are used in the same mix command.

Also, an error is given to the user if the inputs files have different samples rates, or bit resolution. The rules for mixing channels are:

1. MONO files have only one channel while STEREO files have two channels, each processed separately.

2. The maximum volume of a channel is the maximum sample value found over the entire sample file.

3. The mix of all the sound sources is the sum of all the sound sources after they have been multiplied by the relative gain for the source given in the command line. These values are scaled so that the maximum value is appropriate for the output bit resolution. For example, if the bit resolution is 8 bits, then the maximum value in the result of the mix is scaled to 255 and all other values in the sound mix are scaled by this same amount.

Make the appropriate changes to the makefile.

Part H) (100 points) Write a shell script using bash that uses the programs you have written to output a dtmf tone of the specified length. An example command line for running the program is:

gendtmf 5 3.5

This command would generate the tone when the number 5 is pressed on the phone for 3.5 seconds and output the resulting sound file to standard output.

Part I) (100 points) Write a shell script or a C program that takes three parameters of the following form:

gendtmf2 500 250 5155554141

In this example, the output is in the correct format with each of the DTMF tones lasting 500 ms with a 250 ms between each tone.

Modify the makefile appropriately.

Part J) (150 points) Write a program in Java that reads a sound sample file and computes the discrete Fourier of the first 256 samples. The program then outputs the value of the transform (real and imaginary) at the highest frequency and the lowest frequency. For this part of the assignment you must use JNI to call a C function to read the sample file and then another JNI function to compute the discrete Fourier transform. If you do not use JNI to perform these operations, you will receive no credit for this part of the assignment.

Modify the makefile appropriately.

Part K) (300 points) Write a Java GUI program that allows the user to input a mono sound sample file using the FileDialog control. The main window consists of a button to load a new sound file, (which causes the FileDialog to appear) two displays that show the actual sound signal, and the magnitude of the Fourier transform of the 256 samples starting at sample x, and a slider that specifies x. As the slider is moved, the Fourier transform display as well as the signal display is changed appropriately. The signal display of samples shows the same 256 samples used to compute the Fourier transform. Students should feel free to make this GUI as cool as they want to and add any functionality they wish. This only describes the minimum to do for this part of the assignment. Although you may write this entire part of the assignment in Java, the use of a native method to compute the Fourier transform may speed your display.

This part of the program will be tested on the Linux machines in person. Make sure your GUI displays correctly on these machines.

Part L) (300 points) Write an sound effects processor that provides various types of reverb for singers. The reverb may be very short to long echos that could be used for sound effects. There will be a brief lecture on how to algorithmically process a sample file in order to produce the desired reverb effect. Your program will be called reverb and will require three parameters in the following order: predelay, reverb time, and reverb damping.

Part M) (100 points) Write an appropriate makefile for all programs above. Also include a target called all (which should also be the default target) that creates everything. Also include a target called clean that remove all object files, and a target called cleandist that remove all auto generated files created.

Part N) (OPTIONAL) Write a program called dtmf that reads a sample file and outputs the sequence of dtmf buttons pushed based on the contents of the sampled data. The sampled audio resides in a file defined by the MONO format above, and you should read this data from the standard input.

Documentation

Documentation and proper style will be at least 15% of the grade.