Computer Science 229
Homework Project 1 – Part 1
1100 points

C Programming Project. (Fun with digital sound)

Notes

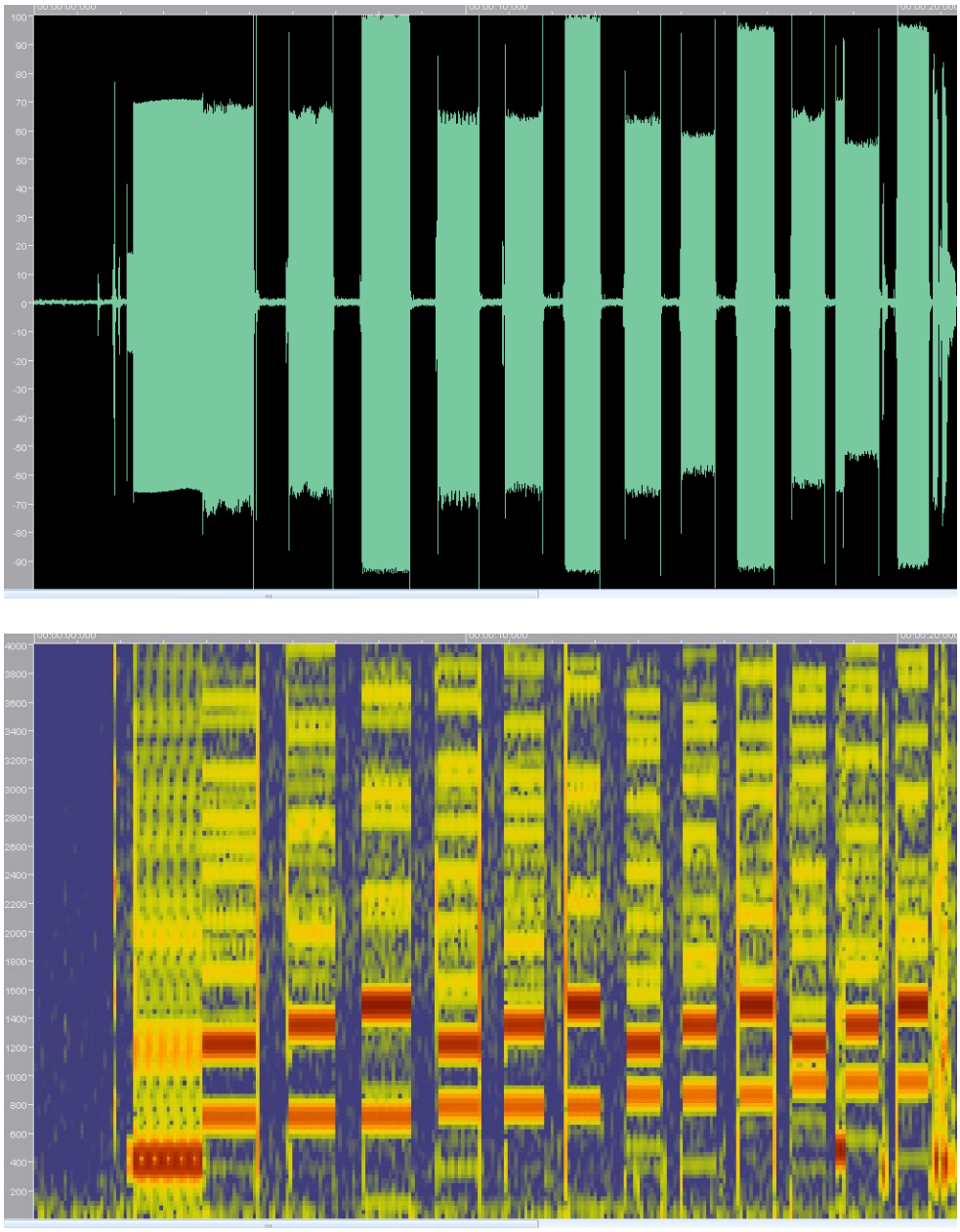
Aspects of this project contain a little math. This is not a class in signal processing nor the mathematics of signal processing. However, in an attempt to create somewhat interesting assignments for both engineers and computer scientists, the programming projects we will do in this class are either 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

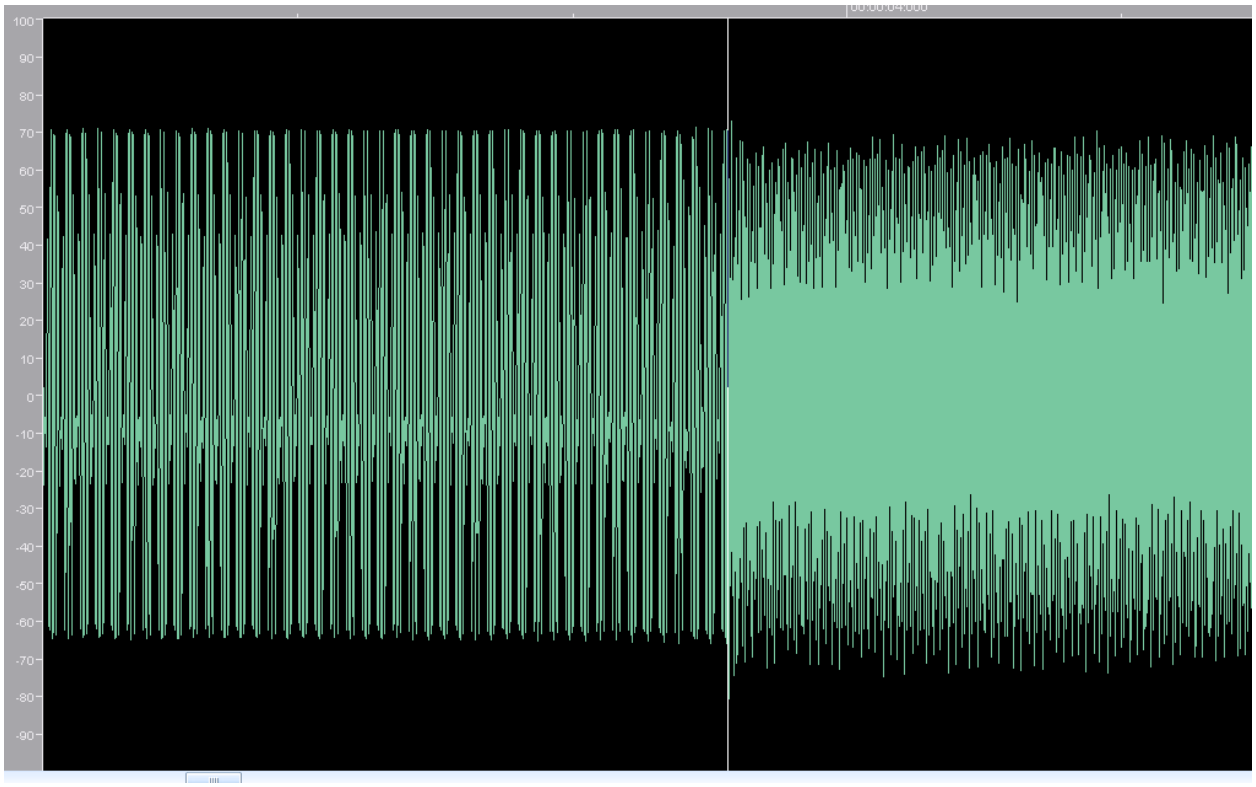
This project utilizes digital sound processing techniques to perform various tasks on small sound samples. Again, you are not responsible for understanding the mathematics behind formulas and algorithms given to you, but you are required to translate these formulas and algorithms into functions and working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

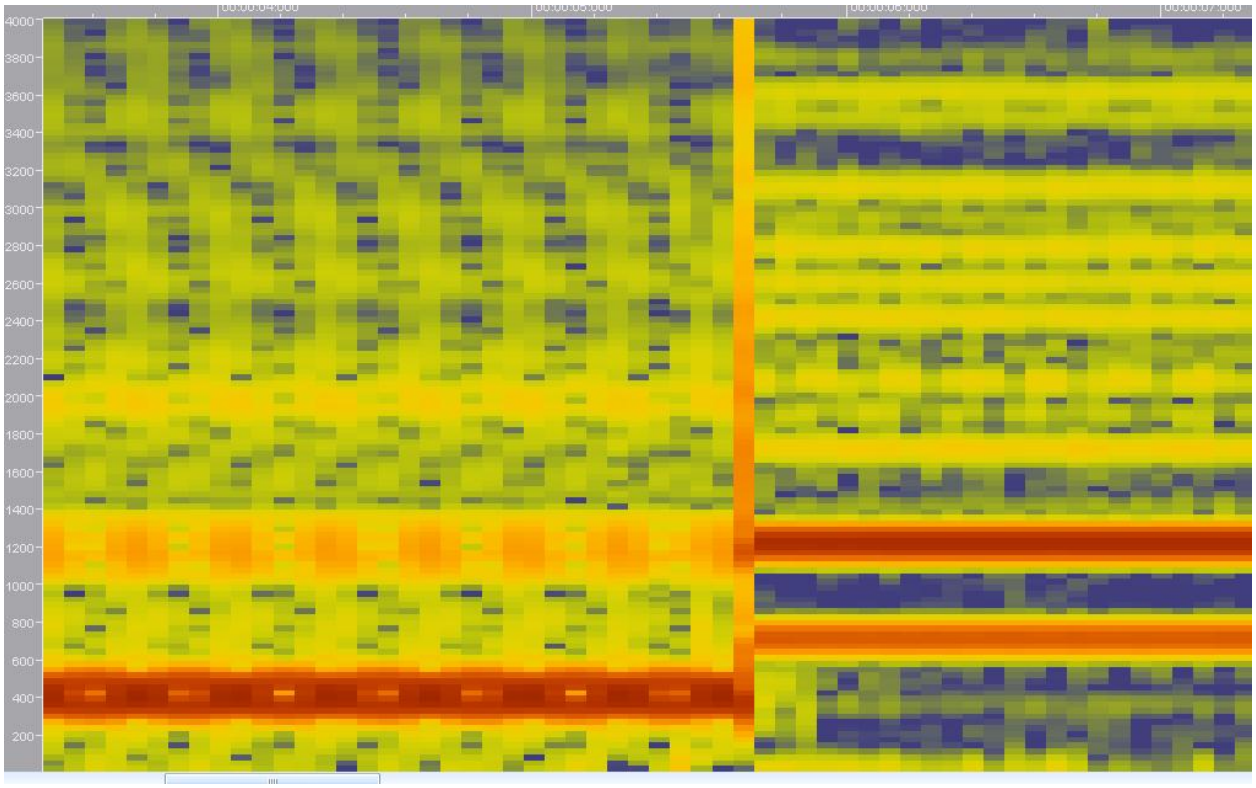
Sound Background Information

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.)

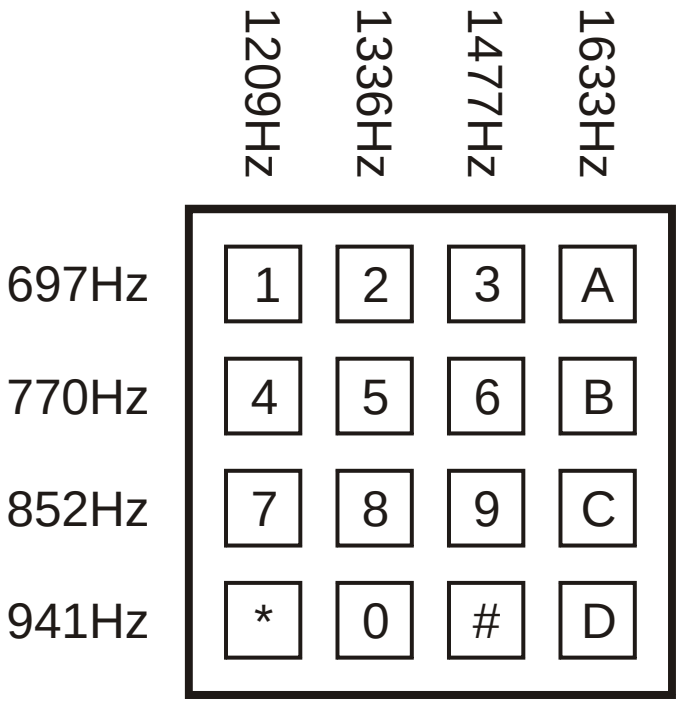
DTMF and your phone

DTMF stands for Dual-Tone Multi-Frequency and is most commonly used in telephone dialing and signaling. In this homework (part 2), you will write C programs to process sound samples and perform various tasks, including encoding 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.

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 algorithms that determine the amount of energy at a particular frequency, 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 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 Formats

In this assignment you will be required to read and write files that contain sound samples. We will use two different formats for storing sound data. The first is a format specific to this class and is not standard. The second is a simple version of the "WAV" format and can be used to play sound files on your computer.

The class specific format is described first, and contains two components: a header part that contains various parameters about the sound data, and the actual data itself.

The header portion is positioned at the beginning of the file before any sound samples. A header consists of the keyword "CS229" 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. (See below.) After all header information is specified, the header section must end with a single line with the word "STARTDATA" in upper or lowercase, followed by a single newline character.

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:

SAMPLE_RATE <sample rate> required.
SAMPLES <number of samples in file> optional
CHANNELS <STEREO | MONO> optional
BITRES <number of bits in a sound value> required

If the SAMPLES 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 most significant byte to least significant byte. (little endian) 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 8 through 15 and the second byte represents bits 0 through 7. 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. Remember, this data is written in binary, not ascii digits.

The second format is a restricted version of the WAV format. This format is described here:

http://www.sonicspot.com/guide/wavefiles.html

Note that your program will only be required to read and write wave files with a "Compression Code" of 1 (PCM/Uncompressed), 1 or 2 channels only, and 8 or 16 byte data only. You will also only need to read or write correctly the "fmt" chunk and the "data" chunk. You are not required to implement any other chunks.

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 formats 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) (400 points) Write a c program that outputs sine wave a specified frequency, format, and duration.

- The output is written to "stdout"
- The executable is named "gensine"
- The parameters are given to the program through command line arguments. The first argument gives the frequency of the sine wave, and the second argument gives the format in which output the sound file data. The format is either "WAV" or "CS229". The third parameter gives the duration of the sine wave in seconds.

  example:

  gensine 440 WAV 1.5
  Outputs a sound file in WAV format of a sine wave of 440 hertz for 1.5 seconds.

- The sound samples are 8-bit MONO
- The sample rate is 8000 samples per second.

Part B) (300 points) Write a C program called "info" that checks if a sound file (of either format) 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 or invalid files.

Part C) (300 points) Write a program called "cs229towav" that converts sound file data in cs229 format to wav format. If there are no command line arguments then the input is read from stdin and the output is written to stdout. If there are command line arguments, then there must be exactly two with the first specifying the input file and the second specifying the output file. Handle all error conditions appropriately. No conversion should be made if there are format errors in the input file.

Part D) (100 points) Write a makefile for the programs in Parts A-C. Be sure that the single command make will create all executable files. Include in your makefile a "clean" target that removes all object and executable files. Your makefile must also only recompile or link source code of any file only when it is necessary.

Documentation

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