# EEN118 LAB ELEVEN

Digital Audio Processing is fundamentally a matter of manipluating a vast quantity of numbers in an array. Being able to record your own voice and hear the results just makes it a little more interesting.

*1.* Revolution 9.

Make sure your equipment is working: write a nice little program that captures the sound of you speaking or singing or humming or whatever, and then plays it back to you. Now make it play back the captured sound in reverse, to see if there are any secret messages hidden in what you said.

*2.* Make the computer hum.

Create an empty sound big enough for a few seconds of music, and with a reasonably high sample rate, something like 20,000 should do it. Look up the correct frequency for middle C, work out the corresponding period (i.e. 1/frequency = time taken for one complete cycle), then work out how many samples would occupy that much time. Fill your array with a repeating sine wave of just the right kind to produce middle C when played.

Let's say you worked out that S samples are needed for one cycle. The sine function goes through one complete cycle when its input angle ranges from 0 to 2π, so you want to scale the input so that S is reduced to 2π, 2S is reduced to 4π, and so on.

Remember also that the sound array must store the sample values as short ints. That means that a waveform with a range of values from -32767 to +32767 would be the loudest thing possible; a range of -1 to +1, which the sine function naturally produces, would be inaudible.

The lab guys have a special doohickey to make sure you have got the note right.

*3.* A more interesting note.

A pure sine wave sounds very sickly and soft and dull. Add some overtones to make it more lively. If you have been generating samples from this formula

```
y = loudness * sin(scale*t)
```

then `y = loudness * sin(scale*t + sin(3*scale*t)/3)`

will sound a bit sharper, and adding more terms of this form:

```
sin(5*scale*t)/5), sin(7*scale*t)/7), .....
```

even more. You may remember this series is from the Fourier analysis of a square wave.

*4.* Create a tune.

Put 8 different notes together (one after the other) to play a scale. Or at least play a few well chosen notes in sequence. Or if you are music-capable, generate a little tune and make it play.

*5.* Pirates.

Make your program read a smallish snippet from an existing music file (it must be in the .WAV format), and play it to you. Maybe you can also play it backwards, to see if it sounds any better that way. A link to free “CD ripping” software is provided on the web page.

*6.* Music Search.

Make a useful little program. When you are trying to find a particular tune or song from all the thousands you have pirated, it can be a very slow process listening to the beginning of each one, until you can tell it isn't the one you want, and skipping to the next. Many songs start very slowly and don't get to the recognisable parts for a long time.

Wouldn't it be good if you had something that would automatically play the second 5 seconds of a song (to skip the silence at the very beginning), followed by the middle 5 seconds, followed by the next-to-last 5 seconds (to skip the silence at the end)? You could probably recognise anything you know if you heard those 15 seconds.

Make it happen.

## For Reference:

The note called “Middle A” is usually taken as the reference for tuning. It has a frequency of 440Hz, which means that one complete waveform including both the positive and negative parts, repeats 440 times per second. Alternatively, it also means that one complete waveform last for 2.272727mS.

In modern western music, the smallest interval between two notes is called a Semi-Tone. If the frequency of one note is $f$, then the frequency of the note one semi-tone higher than it is $f$ times the twelfth root of two. Or in C++, it is f*pow(2.0, 1.0/12.0). Maybe that sounds complicated, but all you need to know is that the twelfth root of 2 is 1.05946, so the note above middle A has a frequency of 466Hz, the next note above that is 494Hz, the next note above that is 523Hz, and so on for ever. To go down by one note, just divide the frequency by 1.05946.

All of that means that if you go up from any starting note by twelve semi-tones, you will arrive at another note with exactly twice the original's frequency. The interval of 12 semi-tones is called an Octave. The names of notes are repeated after an interval of one octave, so the note whose frequency is 880Hz is called High A, and the note whose frequency is 220Hz is called Low A. The names of intermediate notes are a little less logical. This is how they go:

| Name, X | A | A# | B | C | C# | D | D# | E | F | F# | G | G# |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Middle X frequency | 440 | 466 | 494 | 523 | 554 | 587 | 622 | 659 | 698 | 740 | 784 | 830 |
| Alternate name | | Bb | | | Db | | Eb | | | Gb | | Ab |

The table shows the frequencies for all the notes in the octave starting from middle A. The # sign is pronounced "sharp", and the b sign (it is supposed to be a little more pointed than the letter b, but I can’t find the right symbol with this dreadful word processor) is pronounced "flat". The sharps and flats are the black notes on a piano.

We westerners have been subtly trained to prefer certain musical conventions. If you play all 12 notes of an octave in sequence, it sounds somehow wrong. The musical scales that we are familiar with skip certain notes, and have an irregular pattern that sounds right to us. The correct sequence of notes for the "A Major Scale" are A, B, C#, D, E, F#, G#, A. The same pattern applies to all "major" scales; the intervals between successive notes (in semi-tones) are

2, 2, 1, 2, 2, 2, 1.

so D Major would consist of the notes D, E, F#, G, A, B, C#, D. Get the idea? If you want to play a simple scale that sounds right, just follow that pattern of frequency changes.

The sampling rate used when creating CDs is 44,100 samples per second. Unless your computer has really good sound equipment, that is much higher than you need. FM radio uses 22,050 samples per second.