This was a term-long school assignment, running during fall of 2015. It was an opportunity to explore a wide range of concepts, many of which were completely new to me.

The complete program allowed a user to open, create, edit and save audio files. Within the application you can view the waveform and frequencies, and select time ranges, and then cut, copy and paste samples. With some samples selected you can amplify them, reverse them, and even change the pitch for some fun effects. Overall you can change sampling rate and bit rate too.

By the end of this project I had my first programming experience in:

Writing a file reader/writer matching a specification. In this case for a wave file. It was really satisfying to hear my written wave files playing back files in another media player!

Using the low level Win32 multimedia functions to play and record waves. The WaveIn\* and WaveOut\* functions in winmm.dll. It’s so much more empowering to be able to work at a lower level when a given high-level black-box feature is not quite what you need.

Accessing external DLL functions from within C# code by importing them and translating the arguments. Learning how to interact with native code through C# imports was a pain in the butt, and also a positive learning experience.

Using C# itself, all of which was self-taught, and accessing the Window’s clipboard.

These were just the programming tasks though. On a more fundamental level, I learned:

The Discrete Fourier transform, forward and inverse. There was a fun question on the final exam for this course where if you had a black-box function that performed a forward Fourier transform, but no inverse transform, could you use it to convert frequency domain information back to time? How? I’ll just leave it unanswered here!

Filtering audio and the problems and trade-offs of different methods. We talked about a variety of methods, but what I ended up including was filtering by Fourier transform, convolution, and infinite impulse response. It’s so cool—and honestly a little mind boggling—that with the impulse response filters you can take as few as three samples and still do a decent approximation of a highpass or lowpass filter.

Windowing samples. I understand the need for windowing because Fourier’s equation assumes that a given set of samples is periodic. This can be problematic when viewing a small portion of a larger wave where the start and end don’t match (i.e. a high frequency is assumed). Windowing tries to bring the ends close and reduce surprise frequencies, but for the purposes of this program I just used windowing on the Fourier visualization.

Time and pitch shifting. This was an extra I added just for giggles. You can perform pitch-shifting of audio samples by playing them at slower or faster rates. Regrettably my computer won’t let me play a 22,050Hz sample at 21,200Hz, so we do something a little more creative. You can achieve the same pitch shifting effect—and keep your same sample rate—by a 3-step process:

1. Insert a number N samples between every sample you currently have. (these can be anything but might as well be equal to what sample they proceed after.
2. Low-pass filter your samples to turn the stair-step waves created by step 1 back into smooth curves
3. Keep only every D sample in your samples and discard the rest.

The ratio (N/D) is the ratio of speed (and pitch) adjustment you’ve just applied. Cool eh?

The project is hosted on GitHub, with a small description of how to use it.