wavfile: A Simple Sound Library

`wavfile` is a simple sound library for use in CSE 20211. This library allows you to generate arbitrary sound waveforms in an array, then write them out to a standard WAV format file, which can then be played back by almost any kind of computer.

Note: As a courtesy to others in the course or the lab, please use headphones when working with sounds.

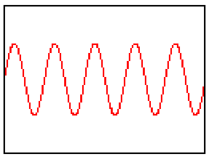
Digital Sound Primer

A computer generates digital sounds by creating a digital **waveform**. A waveform is simply a sequence of digital values that describe how a speaker is to be physically pulsed by an digital-to-analog (DAC) converter. Positive values in the waveform indicate the speaker is to be pushed slightly outward, creating positive air pressure, while negative values indicate the speaker is to be pulled slightly inward, creating negative air pressure.

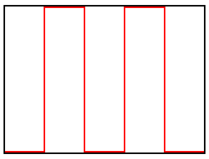
The **sampling rate** of a digital sound indicates how many digital values are used per second of playback. The standard for CD-quality sound is a sampling rate of 44.1KHz. The **volume** of a digital sound is simply the **amplitude** of the waveform: bigger changes mean louder sounds.

The character and quality of a sound is entirely described by the shape of its waveform. Try clicking on the waveforms below to see what they sound like. A sine wave is smooth and open, like a flute. A square wave is piercing, like a smoke alarm. A triangle wave sounds rather brassy. The random wave sounds like white noise.

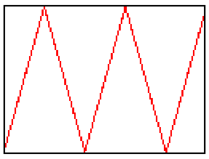
Sine Wave



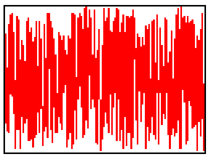
Square Wave



Triangle Wave



Random Wave

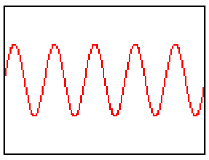


A sound with a regular pattern has a fundamental **frequency** which is the number of peaks in the waveform per second. The sine, square, and triangle waves above all have a frequency of 440Hz, which is a concert-A pitch. (Note also that the frequency is not the same as the sampling rate.)

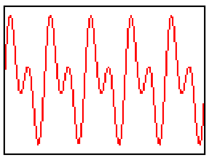
Most sounds in the real world are not as clean and simple as the waveforms above. Instead, they are a sum of multiple waveforms at different frequencies. Musical instruments tend to produce a strong waveform at the pitch actually played. This is known as the **fundamental frequency**. At the same time, they also produce quieter sounds at integer multiples of the fundamental frequency, known has **harmonics**.

For example, click on the three sounds below. The first is a sine wave at 440Hz. The second adds a harmonic at 880Hz. The third adds another harmonic at 1760 Hz. If you listen carefully, you will see that the sounds with more harmonics are more pleasant and more realistic. (I think it sounds like a pipe organ.)

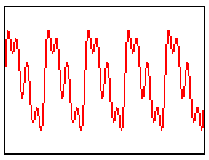
Fundamental Frequency



Fundamental + 1 Harmonic



Fundamental + 2 Harmonics



Getting Started

It is very easy to generate waveforms like those above to create digital sounds and (eventually) music. To get started, download the following files:

- wavfile.h
- wavfile.c
- example.c

And compile them together as follows:

```
gcc example.c wavfile.c -o example -lm
```

Then, run the example program:

```
./example
```

The program creates a file called `sound.wav`. To play back the file on Linux, use the `play` command:

```
play sound.wav
```

How it Works

The sound library only has three functions:

```
FILE * wavfile_open( const char *filename );
```

`wavfile_open` creates a new file whose name is given by the first argument. It automatically puts the standard WAV header into the file for you. If successful, it returns a pointer to a `FILE` object. If unsucessful, it returns null.

```
void wavfile_write( FILE *file, short data[], int length );
```

`wavfile_write` writes data to an open file. The first argument must be a pointer to a `FILE` object returned by `wavfile_open`. The second argument is an array of waveform data, and the third argument is the number of samples to write. This function may be called multiple times to add more sounds to an open wavfile.

```
void wavfile_close( FILE * file );
```

`wavfile_close` completes writing to a wavfile. It is required to call `wavfile_close` when your sound is complete, otherwise the file will not be usable.

```
#define WAVFILE_SAMPLES_PER_SECOND 44100
```

Finally, the header contains a constant `WAVFILE_SAMPLES_PER_SECOND` which indicates how many samples are in a waveform per second of playback.

Example Program

The program `example.c` creates a simple wavfile that plays a sine-wave for one second. It works as follows:

First, an array of `short`s is created to hold a waveform that will last for two seconds:

```
const int NUM_SAMPLES = WAVFILE_SAMPLES_PER_SECOND*2;

short waveform[NUM_SAMPLES];
```

For clarity, we define a few variables to indicate the frequency and volume of the sound. 440Hz is a concert A pitch, and the volume of the waveform is simply the amplitude, which can be up to 32768.

```
double frequency = 440.0;

int volume = 32000;
```

Then, we fill the array with a sine wave of the desired frequency:

```
int i;

for(i=0;i<NUM_SAMPLES;i++) {

    double t = (double) i / WAVFILE_SAMPLES_PER_SECOND;

    waveform[i] = volume*sin(frequency*t*2*M_PI);

}
```

Finally, we use the wavfile library to write out the waveform to a file:

```
FILE * f = wavfile_open("sound.wav");

```

```
wavfile_write(f,waveform,length);

wavfile_close(f);
```

Using the simple sound library, writing out the sound file is easy. All of the challenge lies in constructing a waveform that plays the desired sound.