University of Connecticut
ECE1401

Homework 5
Due April 10, 2024

Objective

Your task is to read a data file with sound data and scale the amplitude and then create a.WAV sound file with the slower audio. WAV files are a Microsoft sound file that stores data samples as raw data – in other words, there is no compression of the data. I have provided you with a test datafile that has 16-bit samples recorded at 22050 samples/second. Unlike the ADC examples that were discussed in lecture, these samples have negative values – i.e. they go from -32768 to 32767 instead of from 0 to 65535. This allows the samples to represent negative voltages without doing any offsets.

Provided Code

I have provided you with the following skeleton `main` function

```
int main(int argc, char **argv)
{
    if (argc != 4) {
        printf("%s datafile wavfile scale\n", argv[0]);
        exit(-1);
    }
    char* datafile = argv[1];
    char* wavfile = argv[2];
    double scale = atof(argv[3]);

    int16_t* sounddata;
    // TODO allocate space for 1000000 int16_t's
    // TODO open the datafile and read the data into sounddata
    // TODO iterate through the array and scale the values

    wavdata_t wav;
    // TODO fill in the wav struct

    write_wav(wavfile, &wav);
}
```

Writing the code

The code takes in three arguments – an input datafile, the name of a .WAV file to write the data with the scaled signal, and a scale value. You will need to read all the data from datafile into an array. You can assume that the file is no bigger than 1000000 sound samples long – that means you should allocate an `sounddata` array that is big enough 1000000 sound samples where each sample is a `int16_t`. You should then read the entire file into `sounddata` with one `read` of 1000000 `int16_t`’s. However, the `read` may return less bytes than that. Keep track of the number of bytes that the `read` returns, since you can use to calculate how many samples you read into the array - each sample is 2 bytes long (the size of an `int16_t` type). In this case, it will be easier to use `open` and `read` instead of `fopen` and `fread`.

Once you figure out the number of samples in `sounddata`, you can iterate through the samples in the `sounddata` array and multiply the values by `scale`. You just have to make sure that the resulting values doesn't exceed 32767 or go below -32768 which are the bounds of an `int16_t` type.

I have provided you with a `write_wav` function in `wav.c` that helps write the data to a .WAV file. However, before you call `write_wav`, you need to fill in the `wav` data structure.

The `wavdata_t` struct is defined as follows:

```
typedef struct {
    uint16_t nchannels;
    uint16_t bits_per_sample;
    uint32_t sample_rate;
    uint32_t datasize;
    int16_t *data;
} wavdata_t;
```

For the provided testfile, `nchannels`=1, `bits_per_sample`=16, and `sample_rate`=22050. `datasize` is the number of bytes in the data set, which is equal to the number of bytes you read from the datafile. `data` is a pointer to an array that holds the data samples.

Once `wav` data structure is filled in, you can call `write_wav`.

Testing the code

I have included an `expected.5.wav`and an `expected2.wav` file that contains the expected output for the following runs of the code. You can compare your output with the expected output using the `diff` command as shown below. If the files are exactly the same, then the `diff` command will not show any differences.

```
$ gcc -o hw5 hw5.c wav.c
$ ./hw5 datafile my.wav 2
$ diff my.wav expected2.wav
$ ./hw5 datafile my.wav 0.5
$ diff my.wav expected.5.wav
```

You should be able to play your output .WAV file and hear the change in volume in the audio.

IMPORTANT: make sure that you get no differences with your output.