# Assignment #2

CS 245, Spring 2018

*Due Thursday, January 25*

I will give you the header file `AudioData.h`, which declares a class named `AudioData` as well as some helper functions, used for manipulating audio data in floating point form. The interface (public and private) of this class is

```
class AudioData {
  public:
    AudioData(unsigned nframes, unsigned R=44100, unsigned nchannels=1);
    AudioData(const char *fname);
    float sample(unsigned frame, unsigned channel=0) const;
    float& sample(unsigned frame, unsigned channel=0);
    float* data(void)             { return &fdata[0]; }
    const float* data(void) const { return &fdata[0]; }
    unsigned frames(void) const   { return frame_count; }
    unsigned rate(void) const     { return sampling_rate; }
    unsigned channels(void) const { return channel_count; }
  private:
    std::vector<float> fdata;
    unsigned frame_count,
             sampling_rate,
             channel_count;
};

void normalize(AudioData &ad, float dB=0);
bool waveWrite(const char *fname, const AudioData &ad, unsigned bits=16);
```

(the standard header file `vector` has been included). The details of these functions are given below. In this assignment you are to implement all but two of the functions in the class interface. Do **not** implement the second constructor `AudioData(fname)` and the `waveWrite` function. You will do this in the next assignment.

`AudioData(nframes,R,nchannels)` — (class constructor) creates an audio data object using floating point data with `nframes` frames, `nchannels` audio channels, and with sampling rate $R$.

`AudioData(fname)` — (class constructor) creates an audio data object by reading the contents of the WAVE file named `fname`. Do **not** implement this function in this assignment.

`sample(frame,channel)` — returns the value of the specified channel within the specified frame of the audio data. There are two versions of this function: one for retrieving the sample value, and one for setting the sample value.

`AudioData::data()` — returns a pointer to the floating point array that stores the audio data. For data that uses more than one audio channel, the channel data is interleaved:

| frame #0 | | | frame #1 | | | ... |
|---|---|---|---|---|---|---|
| channel #0 | channel #1 | ... | channel #0 | channel #1 | ... | ... |

[Implemented]

`AudioData::frames()` — returns the number of frames in the audio data. [Implemented]

`AudioData::rate()` — returns the sampling rate for the audio data. [Implemented]

`AudioData::channels()` — returns the number of channels in the audio data. [Implemented]

`normalize(ad,dB)` — normalizes the audio data `ad` to the specified decibel level `dB`. Normalization should first remove any DC offset from the data. In the case where more than one channel is used, each channel should have the DC offset removed **separately** from each channel; i.e., each channel has its own DC offset that should be removed. After the DC offset is removed, the data values should be scaled so that the largest (absolute) value of the audio data has the specified decibel value relative to unity. That is, 0 dB corresponds to a maximum (absolute) value of 1. Negative decibel values correspond to a maximum that is less than 1, and positive decibel values correspond to a maximum that is greater than 1. Unlike the removal of the DC offset, the maximum value applies to **all data,** regardless of the number of channels; that is, the channels are *not* treated independently (thus it can happen, for example, that the left channel has a larger maximum value than the right channel). Data values should **not** be clipped.

`waveWrite(fname,ad,bits)` — write a WAVE file. Do **not** implement this function in this assignment.

In your implementation of the `AudioData` package, you must assume that the `AudioData.h` header file is *exactly* as stated above. You may **not** add or remove any functions.

For this assignment, you are to submit a single source file named `AudioData.cpp`. You may include only the header file `AudioData.h`, as well as any *standard* C++ header file.