Assignment #3

CS 245, Spring 2018

Due Thursday, February 1

In this assignment, you will implement the remaining functionality of the `AudioData` package from the previous assignment: reading and writing WAVE files. Recall that the class has declaration (both public and private) and helper function prototypes:

```
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);
```

You are to implement the second constructor (the one that takes a file name as argument), as well as the `waveWrite` function. Here are the details.

`AudioData(fname)` — (class constructor) creates an audio data object by reading the contents of the WAVE file named `fname`. If the file is not a valid WAVE file, a `runtime_error` exception is thrown (this is a standard exception defined in the standard C++ header file `stdexcept`). On success, an audio data object is created whose data is filled in with the WAVE file audio data converted to floating point values in the range $[-1, 1]$.

`waveWrite(fname,ad,bits)` — write a WAVE file. Audio data from the `AudioData` object `ad` is written to a WAVE file named `fname`. The data in `ad` are assumed to be floating point values in the range $[-1, 1]$, and are written to the output file using the bit resolution specified by `bits`. The function caller is responsible for ensuring that the

passed–in data is in the range $[-1, 1]$: no clipping is performed. The function returns `true` on success, and `false` otherwise; e.g., if the value of `bits` is not a recognized value (see below).

WAVE file validation. The `AudioData` constructor should throw an exception if the input file is not a valid WAVE file. Specifically, your implementation should check for valid RIFF, format, and data chunks, as well as the presence of the WAVE tag. For the format chunk, format fields should be checked for valid and consistent values.

Your implementation only needs to be able to read and write 8 bit and 16 bit WAVE files. In addition, you may assume that the data has at most two channels; i.e., either mono or stereo data.

Grading. You may assume that the WAVE file supplied to the `AudioData` constructor is in simplified form: header followed by data. However, you will only receive a maximum grade of 93% if you do so. To receive maximum credit, your code should work even if there are chunks other than the required ones present.

The submission for this assignment consists of a single implementation file, named `Wave.cpp` (note that the name is **not** `AudioData.cpp`). You may only include the `AudioData.h` header file, as well as any *standard* C++ header file. In particular, you may **not** include platform specific files, such as `windows.h`.