CS 3005: Programming in C++

WAV File Output

This is a continuation of the previous assignment. Please read the instructions there and complete the required programming tasks before starting this assignment.

Introduction

A WAV file is a file format that can be used to store sound data. It is actually a specific use of the RIFF file format. The WAV file consists of single RIFF chuck, that contains the “RIFF” header, a “fmt ” sub-chunk and a “data” sub-chunk. The “RIFF” header describes the file format, and the “fmt ” sub-chunk contains information about the audio format, such as the number of channels, and the “data” sub-chunk contains the actual audio data.

RIFF Header

The “RIFF” header has three pieces of information.

- The ID, consisting of the 4 characters “RIFF”
- The Size, a 32-bit integer, which is the number of bytes in the file *after* the size. Or, in other words, the number of bytes in the file minus 8.
- The Format, consisting of the 4 characters “WAVE”.

“fmt ” Sub-chunk

The “fmt ” sub-chunk contains information about the audio format, such as the number of channels, and the sample rate. Each of the properties has a specific size and order. Below we describe each of them.

Property	Type	Size	Description
SubchunkID	char	4	“fmt “
SubchunkSize	int	4	the number of bytes in this subchunk that follow this field. In other words, 16.
AudioFormat	int	2	For PCM formats, the value should be 1.
NumChannels	int	2	The number of channels, 1 for mono, 2 for stereo.
SampleRate	int	4	The number of samples per second. (44100 for “CD Quality”)
ByteRate	int	4	The number of bytes per second. SampleRate * NumChannels * BitsPerSample/8.
BlockAlign	int	2	The number of bytes per sample, including all channels. NumChannels * BitsPerSample/8.
BitsPerSample	int	2	The number of bits per sample, per channel.

We want ByteRate and BlockAlign to be an integers, so we will only allow BitsPerSample to be multiples of 8.

“data” Sub-chunk

This block contains the actual sound data. There is a small header at the beginning of the sub-chunk.

Property	Type	Size	Description
SubchunkID	char	4	“data”
SubchunkSize	int	4	the number of bytes in the file that follow this field. In other words, the number of bytes in the file minus 44.

This header is followed by the actual sound data. The first sample is written for all channels. Followed by the second sample for all channels, etc.

The maximum integer value depends on the bits per sample. If there are 8 bits, then the maximum integer value is 127. This is the value achieved by setting all but the most significant bit in the integer to 1. Likewise, if there are 16 bits, then the maximum integer value is 32767. For 24 and 32 bits per sample the maximum values are 8388607 and 2147483647.

Assignment

In this assignment, you will finish creating a class, `WAVFile` that supports the writing of WAV files.

Programming Requirements

Depends on the `AudioTrack` class. Depends on the `endian_io` module. Depends on the `WAVFile` class started in the previous assignment.

Update `library-audiofiles/WAVFile.{h,cpp}`

`WAVFile` Class

Data Members:

The `WAVFile` class should contain data members to track the following information. These data members should be `protected` or `private`. They are not allowed to be `public`.

- `unsigned int` data subchunk position; The position in the file of the data subchunk. Computed while writing a file, and used to write the subchunk size in the correct location, after it has been determined.

`public` Methods:

- `void writeFile(const std::string& filename, const std::vector<AudioTrack>& tracks);` Opens an output file stream using the `open` method, passes the output file stream to `writeFile`, and uses the `close` method to close the output file stream.
- `void writeFile(std::ostream& output_stream, const std::vector<AudioTrack>& tracks);` Sets the data subchunk position to 0 and writes the RIFF header, the FMT subchunk, then writes the data subchunk header. It then writes the track data, and updates the sizes in the headers. Use the other methods of the class to do this work.

`protected` Methods:

- `void open(const std::string& filename, std::ofstream& output_stream);` Open the file named `filename` with the `output_stream`, in binary mode.
- `void writeRIFFHeader(std::ostream& output_stream);` Write the RIFF header. Put 0 as filler data in the chunk size field.
- `void writeFMTSubchunk(std::ostream& output_stream);` Write the “fmt ” subchunk.
- `void writeDataSubchunkHeader(std::ostream& output_stream);` Write the header to the data subchunk. Put 0 as filler data in the chunk size field. Use `tellp` to record the position at the start of the subchunk in the `data subchunk position` member.
- `void writeOneTrackData(std::ostream& output_stream, const double track_data, int maximum_amplitude, int bytes_per_sample);` Computes the integer value to write to the stream from `track_data` and `maximum_amplitude`. Writes the value in little endian order to `output_stream`.
- `void writeTracks(std::ostream& output_stream, const std::vector<AudioTrack>& tracks);` Write the track data in little endian order. If there are not exactly 2 tracks, or if the two tracks are not the same size, do nothing. Note that the bytes per sample is the bits per sample divided by 8. Also, the maximum amplitude depends on the bits per sample.
- `void writeSizes(std::ostream& output_stream);` Now that the sizes of chunks and file are known, go back to the headers and write the correct values in the size fields. Assumes that the output_stream is now positioned at the end of the file. This will allow us to compute the total file size. Use `tellp()` to find the current stream position. Use `seekp()` to move to the location where the size should be written.
- `void close(std::ofstream& output_stream);` Close `output_stream`, if it is open.

Additional Documentation

- WAVE PCM soundfile format
- std::ostream
- std::ofstream
- namespace
- types
- Member initializer list
- if
- comparison operators
- logical operators
- std::stringstream
- arithmetic operators

- std::vector
- std::string
- #include
- const method
- references
- public/protected/private
- size_t
- classes
- class declaration
- class implementation file
- static_cast
- auto
- for (counted)
- for (range)
- address-of

Grading Instructions

To receive credit for this assignment:

- your code must be pushed to your repository for this class on GitHub
- all unit tests must pass
- all acceptance tests must pass
- all programs must build, run, and execute as described in the assignment descriptions.

Extra Challenges (Not Required)

Describe possible additions and extensions that could be added without breaking the expected functionality. Careful not to give away too many possible exam tasks.