CS 3005: Programming in C++

WAV File Output

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".

Assignment

In this assignment, you will start to create a class, WAVFile that supports the writing of WAV files.

Programming Requirements

Depends on the AudioTrack class.

Create [library-audiofiles/endian_io.{h,cpp}]

Declare the following functions in the header file, and implement them in the cpp file. Some of the functions belong to the <u>little_endian_io</u> namespace, and others to the <u>big_endian_io</u> namespace. Be sure to declare them correctly.

little_endian_io Functions:

- std::ostream& write_word(std::ostream& output_stream, int value, unsigned size); Writes size bytes of
 value to output_stream. Starts with the least significant byte.
- std::ostream& write_4_bytes(std::ostream& output_stream, int value); Uses write_word to write 4 bytes.
- std::ostream& write_3_bytes(std::ostream& output_stream, int value); Uses write_word to write 3 bytes.
- [std::ostream& write_2_bytes(std::ostream& output_stream, int value); Uses write_word to write 2 bytes.
- std::ostream& write_1_bytes(std::ostream& output_stream, int value); Uses write_word to write 1 bytes.

big_endian_io Functions:

• std::ostream& write_string(std::ostream& output_stream, const std::string& value); Writes the string
value to output_stream in order from first to last byte of the string.

Create [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.

- int samples per second; The sampling rate of the data for the wave file.
- int bits per sample; The number of bits per sample in the wave file. Constrained to a multiple of 8 (specifically 8, 16, 24, or 32).

public Methods:

- WAVFile(int samples_per_second, int bits_per_sample); Initializes data members from the parameters to 0 in the initialization part of the constructor. Uses the setter methods to attempt to set the data members from parameters. Note, this means, if samples_per_second is less than 1 then the data member will still be set to 0, or if bits_per_sample is not a multiple of 8, then the data member will be set to 0.
- int getSamplesPerSecond() const; Returns the value of the data member.
- int getBitsPerSample() const; Returns the value of the data member.
- void setSamplesPerSecond(const int samples_per_second); Assigns the value of the data member. If the new value is less than 1, then the method should do nothing.
- void setBitsPerSample(const int bits_per_sample); Assigns the value of the data member. If the new value is not a multiple of 8 (8, 16, 24, or 32), then the method should do nothing.

Update [library-audiofiles/Makefile]

• Add the WAVFile and endian files to the Makefile so the headers are installed and the ... file are added to the library.

Additional Documentation

- <u>WAVE PCM soundfile format</u>
- std::ostream
- <u>std::ofstream</u>
- <u>namespace</u>
- <u>types</u>
- <u>Member initializer list</u>
- <u>if</u>
- <u>comparison operators</u>
- <u>logical operators</u>
- std::stringstream
- arithmetic operators
- <u>std::vector</u>
- <u>std::string</u>
- <u>#include</u>
- <u>const method</u>
- <u>references</u>
- public/protected/private
- <u>size_t</u>
- <u>classes</u>
- <u>class declaration</u>
- class implementation file
- <u>static_cast</u>
- <u>auto</u>
- <u>for (counted)</u>
- for (range)
- <u>address-of</u>

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)

• None at this time