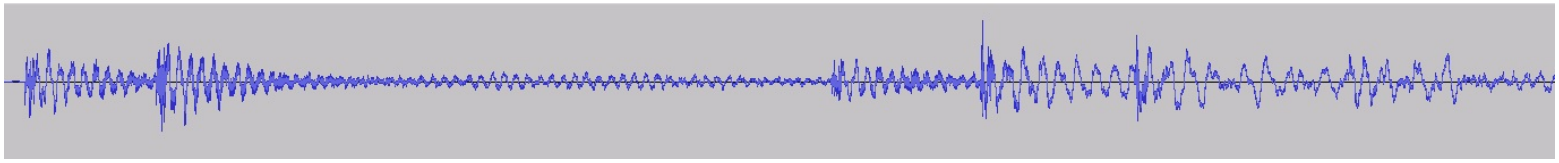
CS 3005: Programming in C++

Audio Track Class

Introduction

A digital audio track is a sequence of values that describe the intensity of sound at different times. For example, this image show the first second of Eddie Van Halen's classic guitar solo "Eruption".



Audio engineers collect and combine audio tracks to make music files, such as those released by artists for your listening pleasure.

To create higher quality audio, the tracks need to record more samples per second. The more samples per second, the more data, and storage, is needed. So, trade offs are made between quality and storage.

Assignment

In this assignment, you will create a class called `AudioTrack` that describes an audio track. It will have the ability to store and retrieve audio track information. An audio track object will be configurable with the number of samples per second (quality) and the number of seconds (duration).

The total number of samples in an audio track is determined by the number of samples per second multiplied by the number of seconds. Anytime an audio track object's samples per second or number of seconds is changed, the size of the audio track will change. Your class will control this by automatically resizing a `vector` to meet the requirements.

In this assignment, you will be creating the first portion of the `library-audiofiles` library. In future assignments, another class will be added to allow you to write audio files that can be played back. There is no program for this assignment.

Programming Requirements

Create `library-audiofiles/AudioTrack.{h,cpp}`

`AudioTrack` Class

Data Members:

The `AudioTrack` 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 audio data in the track.
- `double` seconds; The duration of the audio data in seconds.
- `std::vector<double>` values; The audio data in the form of a sequence of samples. The samples should be initialized to 0. The length of the vector should be sufficient to contain the number of samples specified by the `samples per second` and `seconds` data members.

`public` Methods:

- `AudioTrack();` Initializes data members to zero values.
- `int getSamplesPerSecond() const;` Returns the value of the data member.
- `double getSeconds() const;` Returns the value of the data member.
- `unsigned int getSize() const;` Returns the size of the `values` vector.
- `void setSamplesPerSecond(const int samples_per_second);` Assigns the value of the data member. Resizes the `values` vector to reflect the new size of the audio data. If the new value is less than 1, then the method should do nothing.
- `void setSeconds(const double seconds);` Assigns the value of the data member. Resizes the `values` vector to reflect the new size of the audio data. If the new value is 0 or less, then the method should do nothing.
- `void setSize(const int samples_per_second, const double seconds);` Assigns the values of the data members. Resizes the `values` vector to reflect the new size of the audio data. If either of the new values is less than or equal to 0, then the method should do nothing.
- `bool indexValid(const unsigned int index) const;` Returns whether the parameter is a valid index to the `values` vector.
- `double getValue(const unsigned int index) const;` Returns the sample at the given index of the `values` vector. If the index is invalid, then the method should return `-INFINITY`.
- `void setValue(const unsigned int index, const double value);` Assigns the sample at the given index of the `values` vector to the given value. If the index is invalid, then the method should do nothing.

`protected` Methods:

- `void resizeValues();` Resizes the `values` data member to the appropriate size, determined by the `samples per second` and `seconds` data members. The `values` vector should be zeroed out as well.

Create `library-audiofiles/Makefile`

This file must contain rules such that any of the following commands will build the `libaudiofiles.a` library:

- `make`
- `make all`

This file must contain rules such that the following command will install the `libaudiofiles.a` library into the `lib` directory:

- `make install`

Update `Makefile`

Update the project-level `Makefile` so that `make` and `make all` in the project directory will call `make install` in the `library-audiofiles` directory.

Additional Documentation

- classes
- public/protected/private
- std::vector
- std::vector::resize
- cmath (for INFINITY)

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)

- Add a makefile target to push your code to your repository
  - Non-file (Phony) Targets
- Add git pre-commit hooks to automatically format code before committing
  - Git hooks
  - Clang Formatter