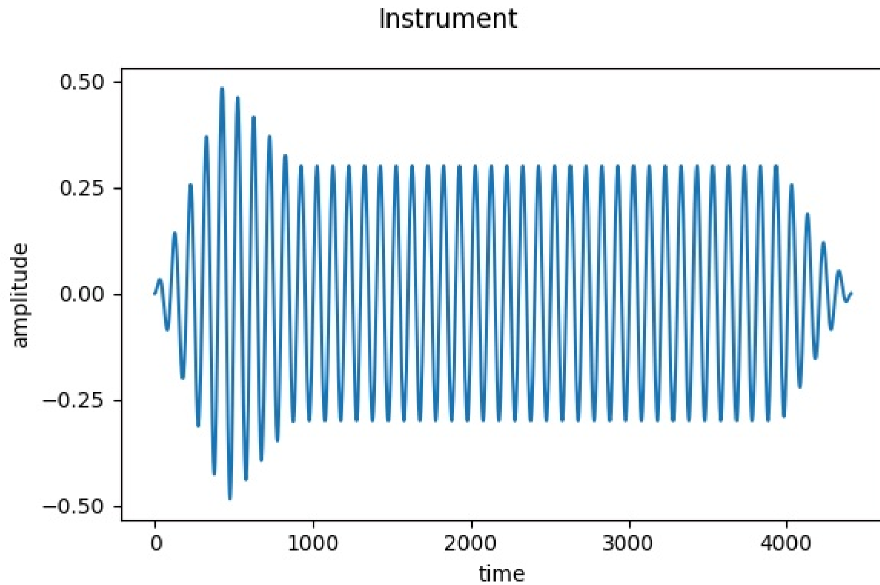
CS 3005: Programming in C++

Instrument Class

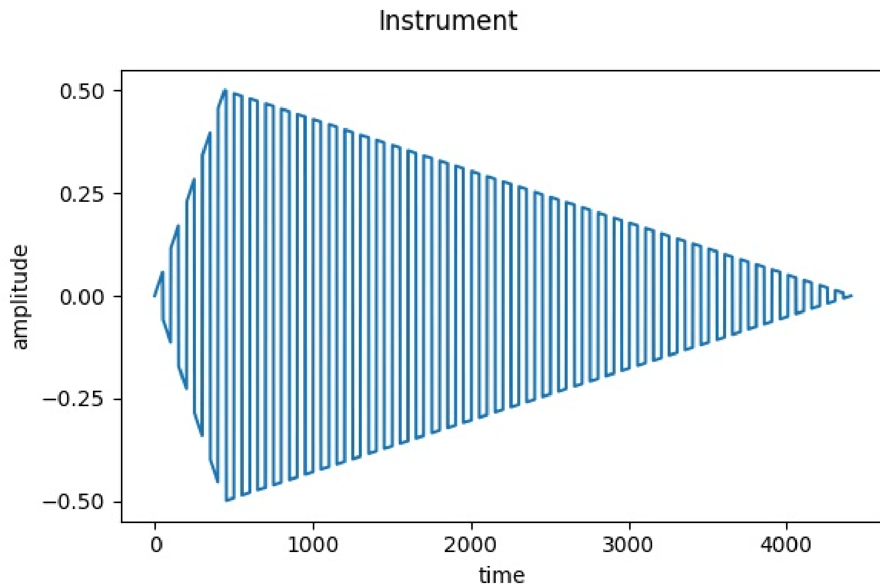
Introduction

We will define an instrument for audio generation as a combination of a waveform and an envelope. The waveform will provide the shape of the oscillating sound waves and the envelope will provide the shape of the amplitude of individual notes played by the instrument.

The first example shows a sine waveform with an ADSR envelope.



The second example shows a square waveform with an AD envelope.



Assignment

In this assignment, you will create a class to represent an instrument. This class will also be capable of rendering a sound for an instrument using its waveform and envelope. The instrument can be used to generate a variety of sounds, with different frequencies and durations.

You will also create a simple program that generates an instrument and saves a sound generated by the instrument to a WAV file.

Sample interactions with the program may look like this:

A sine waveform with an AD envelope

```
$ ./program-instrument-test/instrument_test
Samples/Second: 44100
Seconds: 0.5
Bits/Sample[8,16,24,32]: 32
Waveform style: sine
Envelope style: AD
Maximum amplitude: 1.0
Attack seconds: 0.1
Frequency: 440.0
WAV filename: instrument-sine-AD.wav
```

A square waveform with an ADSR envelope

```
$ ./program-instrument-test/instrument_test
Samples/Second: 44100
Seconds: 0.5
Bits/Sample[8,16,24,32]: 32
Waveform style: square
Envelope style: ADSR
Maximum amplitude: 1.0
Attack seconds: 0.05
Decay seconds: 0.15
Release seconds: 0.005
Sustain amplitude: 0.7
Frequency: 400.0
WAV filename: instrument-square-ADSR.wav
```

Bad waveform or envelope names

```
$ ./program-instrument-test/instrument_test
Samples/Second: 44100
Seconds: 0.5
Bits/Sample[8,16,24,32]: 32
Waveform style: fred
Waveform style 'fred' is not known.
Envelope style: wilma
Maximum amplitude: 1.0
Envelope style 'wilma' is not known.
Frequency: 440.0
Segmentation fault (core dumped)
```

The segmentation fault is because the waveform or envelope are not configured. We will consider this to be acceptable behavior for this test program.

Multiplying AudioTrack Objects

If two audio tracks have the sample number of samples, you can multiply them on a sample by sample basis. For example, if the first audio track has the five values `[0.1, 0.2, 0.3, 0.4, -0.5]` and the second track has the five values `[0.6, 0.7, 0.8, 0.9, 1.0]`, the result will be and audio track object with the values `[0.6, 0.14, 0.24, 0.36, -0.5]`.

It only makes sense to multiple two audio tracks if they have the same sample rate (samples per second) and duration (seconds).

Generating AudioTrack Samples for an Instrument

The envelope for an instrument is used to fill an audio track object, using the envelope's method for generating amplitudes. Note this is a set of positive amplitudes based on the form of the envelope object.

The waveform for an instrument is used to fill a different audio track object, using the waveform's method for generating samples. Note that this is a set of values that oscillate between positive and negative values depending on the shape of the waveform, and the frequency of the sample.

Finally, an instrument's audio track samples are set by multiplying the two audio track objects together.

Creating an Instrument

An instrument is a pairing of an envelope and a waveform. To create an instrument, the envelope and waveform are created first, then they are used to configure the new instrument.

Programming Requirements

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

`AudioTrack` Class

`public` Methods:

- `AudioTrack operator*(const AudioTrack& rhs) const;` This overloaded operator will create a new `AudioTrack` object that is the product of the two `AudioTrack` objects. If the two `AudioTrack` objects are not compatible, the function will return an empty `AudioTrack` object. The resulting `AudioTrack` object has the same sample rate and duration as the first `AudioTrack` object. See the multiplication description above.

Create `library-instrument/Instrument.{h,cpp}`

`Instrument` Class

Data Members:

The `Instrument` 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`.

- `std::string` name for the instrument; Used to identify an instrument in collections of many instruments.
- `std::shared_ptr<Waveform>` waveform for the instrument; Used to generate sound from the instrument.
- `std::shared_ptr<Envelope>` envelope for the instrument; Used to control the amplitude of the sound from the instrument.

`public` Methods:

- `Instrument();` Default constructor for the class. Allows all data members to be default constructed as well.
- `Instrument(const std::string& name, std::shared_ptr<Waveform> waveform, std::shared_ptr<Envelope> envelope);` Constructor for the class. Initializes all data members from input parameters.
- `virtual ~Instrument();` Required for polymorphic classes. Empty body for now.
- `const std::string& getName() const;` Return the data member.
- `std::shared_ptr<Waveform> getWaveform() const;` Return the data member.
- `std::shared_ptr<Envelope> getEnvelope() const;` Return the data member.
- `void setName(const std::string& name);` Update the data member.
- `void setWaveform(std::shared_ptr<Waveform> waveform);` Update the data member.
- `void setEnvelope(std::shared_ptr<Envelope> envelope);` Update the data member.
- `void generateSamples(const double frequency, const double seconds, const int samples_per_second, AudioTrack& track) const;` Fills the `AudioTrack` object with samples generated from the `Instrument` object. See the description above.

Create `library-instrument/Makefile`

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

- `make`
- `make all`

This file must contain rules such that the following command will install the `libinstrument.a` library into the `lib` directory, and all `.h` files to the `include` directory:

- `make install`

Create `library-commands/instrument_test_aux.{h,cpp}`

Functions:

- `std::shared_ptr<Waveform> choose_waveform(ApplicationData& app_data);` Allows the user to choose a "Waveform style: ". If the user's choice is "sine" or "square", then creates a `std::shared_ptr` object with the correct type. The name of the Waveform object created should be the empty string. If the user's choice isn't then, sends a message to the user "Waveform style 'BAD_USER_CHOICE' is not known." In any case, shared pointer to a `Waveform` object is returned. In the error case, the pointer is the null pointer.
- `std::shared_ptr<Envelope> choose_envelope(ApplicationData& app_data);` Allows the user to choose a "Envelope style: " and "Maximum amplitude: ". If the user chooses "ADSR", creates a `std::shared_ptr` to `ADSREnvelope`. If the user chooses "AD", the creates a shared pointer to `ADEnvelope`. If the user choice is anything else, then displays a message for the user, "Envelope style 'BAD_USER_CHOICE' is not known.". In the error case, the shared pointer is set to the null pointer. Returns the shared pointer. The ADSR envelope needs to allow the user to configure "Attack seconds: ", "Decay seconds: ", "Release seconds: ", and "Sustain amplitude: ". The AD envelope needs to allow the user to configure "Attack seconds: ". Either case will set the envelope's maximum amplitude from the user's response.
- `std::shared_ptr<Instrument> create_instrument(ApplicationData& app_data);` Uses other functions to allow the user to choose a waveform and an envelope. Uses them to create an instrument. The instrument's name should be the empty string. The instrument will be created as a shared pointer, and returned.
- `void fill_audio_track_from_instrument(ApplicationData& app_data, std::shared_ptr<Instrument> instrument_ptr);` Fetches the frequency from the application data's `doubleRegister` in slot 0. Assumes the meta data of the audio in the application data has already been configured. Uses the instrument class method to generate samples into the application data's audio track.
- `int instrument_test(ApplicationData& app_data);` Configures the audio track and WAV file using `configure_audio_track_and_wav_file()`. Creates an instrument using `create_instrument()`. Asks the user for a "Frequency: ", and stores it in the application data's `doubleRegister` in slot 0. Calls `fill_audio_track_from_instrument()` to fill the audio track with samples from the instrument. Configures the channels in the application data to have 2 channels. Sets both of the audio tracks from the audio track that was filled from the instrument. Uses `save_wav_file()` to save the data to a file. Returns 0.

Update `library-commands/Makefile`

Add `instrument_test_aux.{h,cpp}` in the appropriate places to add them to the library and install the header file.

Create `program-instrument-test/instrument_test.cpp`

Functions:

- `int main();` Entry point to the instrument test program. Should create an `ApplicationData` and pass it to the `instrument_test` function found in `instrument_test_aux` and return the result of that function call.

Create `program-instrument-test/Makefile`

This file must contain rules such that any of the following commands will build the `instrument_test` program:

- `make`
- `make all`
- `make instrument_test`

Create `program-instrument-test/.gitignore`

The file `program-instrument-test/.gitignore` needs to store one line of text:

```
instrument_test
```

Update `Makefile`

Update the project-level `Makefile` so that `make` and `make all` in the project directory will call `make install` in the `library-instrument` directory, and `make` in the `program-instrument-test` directory.

Additional Documentation

TBA

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)

TBA