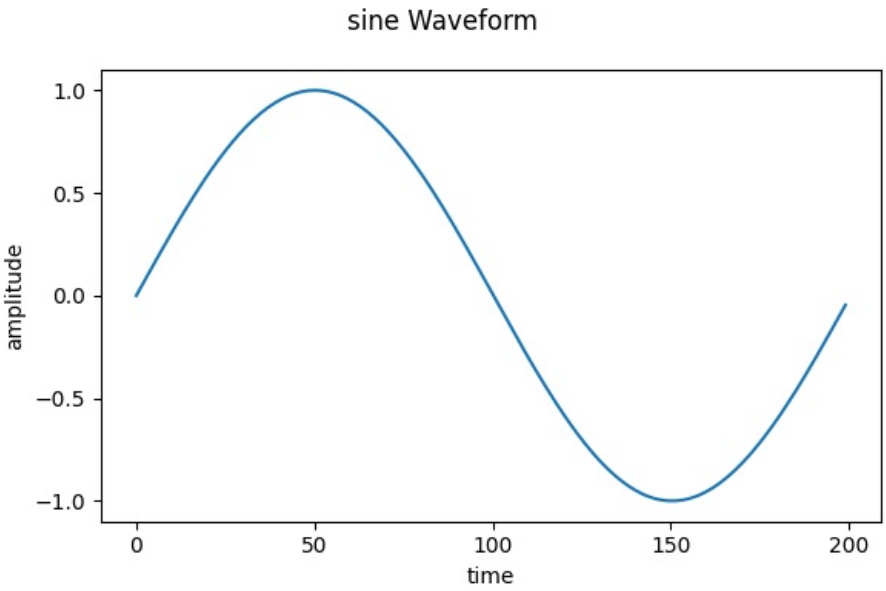
CS 3005: Programming in C++

Waveform Classes

Introduction

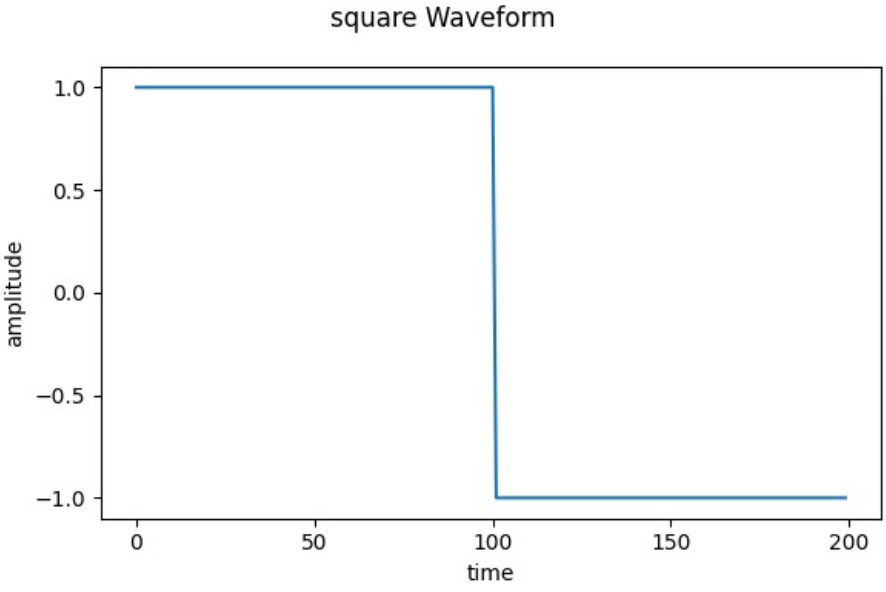
Sound is sent through the air as compressions and expansions of the density of the air molecules. A single pitch sound comes from a cyclic compression/decompression at a specific frequency. For example, Middle C on a piano keyboard produces sound at approximately 261.6258 Hz (Hz = cycles per second).

When working with digital representations of sound, we record the increase or decrease in the air density, as the signal. For example, the following image shows a cycle of a sine wave. We allow the amplitude to range between 1.0 and -1.0, indicating the most compression and the most decompression.



Notice that the amplitude increases and decreases smoothly. You can hear the smooth sound produced by this waveform in the audio track. If we reduce the amplitude of the signal, the sound becomes more quiet. Increasing the frequency of the sine wave will produce a higher pitched sound, and decreasing the frequency will produce a lower pitched sound.

We can also change the shape of the signal to produce a different sound. This is a square wave. It abruptly changes between the maximum and minimum amplitude, and has a harsher sound.



Audio designers will try many different shapes to create different sounds. The amplitude is used to control

the volume of the sound, and the frequency is used to control the pitch of the sound.

Assignment

In this assignment, you will create a class hierarchy to represent the functionality of a waveform, including specialization for the sine and square waves.

The `Waveform` class will be a base class. `SineWaveform` and `SquareWaveform` will inherit from it. The base class will be responsible for storing the amplitude of the wave. It will also provide the functionality to compute the wave's position in a cycle, given the time since the beginning of the wave. It will also provide the functionality of generating all samples for a waveform over a given period of time, storing the samples in an `AudioTrack` object. The derived classes will provide the functionality capable of producing the specific shape of the waveform.

You will also create a simple program that writes a WAV file after the user selects the waveform, amplitude and frequency, and the duration of the audio track.

A sample interaction with the program may look like this:

```
$ ./program-waveform-test/waveform_test
Samples/Second: 44100
Seconds: 2.5
Bits/Sample[8,16,24,32]: 16
Left Channel
Waveform style: sine
Amplitude: 1.0
Frequency: 265
Right Channel
Waveform style: square
Amplitude: 0.5
Frequency: 300
WAV filename: foo.wav
$ ls -l foo.wav
-rw-rw-r-- 1 cgl cgl 441044 Oct 10 14:41 foo.wav
```

This example demonstrates a user giving an invalid waveform style.

```
$ ./program-waveform-test/waveform_test
Samples/Second: 8000
Seconds: 1.2
Bits/Sample[8,16,24,32]: 24
Left Channel
Waveform style: sawtooth
Amplitude: 1.0
Frequency: 400
Waveform style 'sawtooth' is not known.
Right Channel
Waveform style: sine
Amplitude: 1.0
Frequency: 300
WAV filename: bar.wav
$ ls -l bar.wav
-rw-rw-r-- 1 cgl cgl 57644 Oct 10 14:45 bar.wav
```

Computing Position in Waveform (Angle from Sample Number)

The units for frequency are Hertz (Hz) which is the same as (1/Seconds). The units for `samples_per_second` are Samples/Seconds. The units for `sample_number` are Samples. The units of `two_pi` are Radians. The value of `two_pi` is approximately `6.283185307179586476925286766559`.

To compute the angle from the sample number we need to use this formula:

```
angle = two_pi * sample_number * frequency / samples_per_second
```

Note the units are:

```
Radians = Radians * Samples * (1/Seconds) / (Samples/Seconds)
Radians = Radians * Samples * (1/Seconds) * (Seconds/Samples)
Radians = Radians * (Samples/Samples) * (Seconds/Seconds)
Radians = Radians
```

This checks out.

Computing Cycle Position

The cycle position is the fractional part of a cycle within the full position. We want to make this be a number in the range `[0, 1)`.

This is calculated by getting the angle of the full position, then repeatedly subtracting of the angle of a full cycle, until the remaining angle is less than a full cycle. The angle of a full cycle is `two_pi`. Note that this repeated subtraction is the same as taking the modulus of the angle by `two_pi`.

The problem is that the modulus operator (`%`) only works on integers. We are working with floating point numbers. We need to use `std::fmod()` (look in the documentation to find the correct header file) to compute the remaining angle.

After getting the remaining angle, we get the position in the cycle by dividing by `two_pi`.

Programming Requirements

Create `library-waveform/Waveform.{h,cpp}`

`Waveform` Class

Data Members:

The `Waveform` 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 waveform; Used to identify a waveform in collections of many waveforms.
- `std::string` type name for waveform; Used to identify the specialized form of the waveform.
- `double` amplitude; The amplitude of the waveform in the range of 0 to 1.

`public` Methods:

- `Waveform(const std::string& name, const std::string& type_name);` Initializes the two string data members using the parameters, and sets the amplitude to 1.
- `virtual ~Waveform();` Empty function body. This method is required because we have a virtual method.
- `const std::string& getName() const;` Returns the data member.
- `const std::string& getTypeName() const;` Returns the data member.
- `double getAmplitude() const;` Returns the data member.
- `void setName(const std::string& name);` Updates the data member.
- `void setTypeName(const std::string& type_name);` Updates the data member.
- `void setAmplitude(const double amplitude);` Updates the data member, but only if the amplitude is between 0 and 1, inclusive.
- `double computeSampleAngle(const double frequency, const double sample_number, const int samples_per_second) const;` Computes the angle (in radians) that corresponds to the `sample_number`'s position from the beginning of samples. See discussion above for mathematical details.
- `double computeSampleCyclePosition(const double frequency, const double sample_number, const int samples_per_second) const;` Computes the position in the cycle for `sample_number`. See discussion above for mathematical details.
- `virtual void generateSamples(const double frequency, const double seconds, const int samples_per_second, AudioTrack& track) const;` Sets the size of `track`, then for every `sample_number` in the track, sets the value to the result of `generateOneSample()`.
- `virtual double generateOneSample(const double frequency, const int sample_number, const double samples_per_second) const = 0;` Pure virtual method that must be overridden (specialized) by the subclasses.

Special Note

It is recommended that you define a constant in the `Waveform.cpp` file to use the precise same value for `two_pi` everywhere it is needed.

```
const double two_pi = 6.28318530717958647692528676655900;
```

Create `library-waveform/SineWaveform.{h,cpp}`

`SineWaveform` Class

Publicly inherits from `Waveform`.

Data Members:

The `SineWaveform` class does not define any new data members.

`public` Methods:

- `SineWaveform(const std::string& name);` Constructor chains to the base class constructor. Uses “sine” for the type name.
- `virtual ~SineWaveform();` Empty body, but required.
- `virtual double generateOneSample(const double frequency, const int sample_number, const double samples_per_second) const;` Computes the angle associated with `sample_number`. Computes the sine of that angle. Multiplies the result by the amplitude before returning it. See `std::sin()`.

Create `library-waveform/SquareWaveform.{h,cpp}`

`SquareWaveform` Class

Publicly inherits from `Waveform`.

Data Members:

The `SquareWaveform` class does not define any new data members.

`public` Methods:

- `SquareWaveform(const std::string& name);` Constructor chains to the base class constructor. Uses “square” for the type name.
- `virtual ~SquareWaveform();` Empty body, but required.
- `virtual double generateOneSample(const double frequency, const int sample_number, const double samples_per_second) const;` Computes the cycle position associated with `sample_number`. If it is less than 0.5, returns amplitude. Otherwise, returns -amplitude.

Create `library-waveform/Makefile`

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

- `make`
- `make all`

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

- `make install`

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

Functions:

- `void fill_audio_track_with_waveform(ApplicationData& app_data);` Prompts the user for string “Waveform style: “, double “Amplitude: “, and double “Frequency: “. Based on the waveform style selected (“sine” or “square”), creates a `SineWaveform` or `SquareWaveform` object, and sets the amplitude of the waveform. Then generates samples and stores them in the ApplicationData object’s AudioTrack object. Assumes that the AudioTrack object’s meta data (seconds, and samples per second) are already set correctly. If the user chooses a different waveform style, then the error message is displayed: “Waveform style ‘USER_RESPONSE’ is not known.“, then does not fill the AudioTrack object with data.
- `void fill_channels_with_waveforms(ApplicationData& app_data);` Sets the size of the application data object’s channels to 2, the calls fill_audio_track_with_waveform for each of the channels, storing the

AudioTrack data in the correct channel. Displays “Left Channel” or “Right Channel” for the 0 or 1 channel.
- `int waveform_test(ApplicationData& app_data);` Configures the AudioTrack and WAVFile in the ApplicationData object using the `configure_audio_track_and_wav_file()` function from a previous task. Fills the channels of the application data object using `fill_channels_with_waveforms()`, then saves the wav file with `save_wav_file()` from a previous task. Returns 0.

Update `library-commands/Makefile`

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

Create `program-waveform-test/waveform_test.cpp`

Functions:

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

Create `program-waveform-test/Makefile`

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

- `make`
- `make all`
- `make waveform_test`

Create `program-waveform-test/.gitignore`

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

```
waveform_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-waveform` directory, and `make` in the `program-waveform-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