OBJECT ORIENTED PROGRAMMING

SEMINAR 3

OBJECTIVES

In this seminar we will write a simple application that is able to generate and play basic acoustic waveforms. For this task, we will create several classes and organize them into a class hierarchy. We will also work with static methods and practice operator overloading.

PROPOSED PROBLEM

Sound is a vibration, an acoustic wave that propagates through a transmission medium. When this vibration of the air molecules reaches our ears, they like a powerful convertor that transforms this acoustic signal into an electric impulse that can be understood by the brain.

The basic building block of this application would be an abstract data type that represents a wave. However, different types of waveforms exist, with different characteristics, which result in different sounds. For this project, we'll work with the following waveforms: *sine, square, triangle* and *sawtooth*. One can find out more about each waveform at the following website: https://www.perfectcircuit.com/signal/difference-between-waveforms.

So, we'll begin by designing a base class that stores an acoustic wave: *AcousticWave*. The class should have the following members:

- The frequency of the wave – this encodes the pitch of the sound;
- The amplitude of the sound – this encodes the "volume" of the sound;
- The sample rate – when working with digital devices we cannot directly process the analog signal, so we need to sample it at different equally spaced intervals;
- The actual samples that make up the wave. This should be a dynamically allocated array.

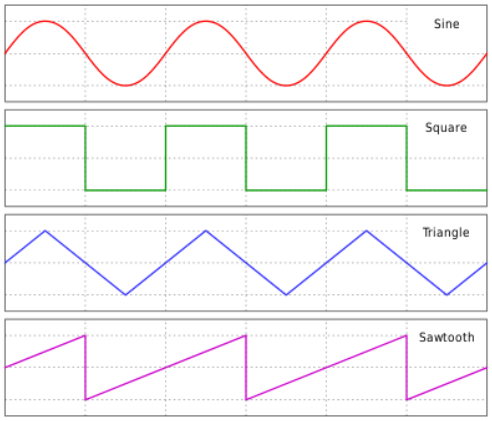

Figure 1. Different types of waveforms. Image source: https://en.wikipedia.org/wiki/Sawtooth_wave

We **should** define a copy constructor, destructor and overload the assignment operator for the *AcousticWave* class. Why is that?

Next, we will define four subclasses of *AcousticWave*, one for each type of wave: *SineWave, SquareWqve, TraingleWave* and *SawtoothWave.*

The base class *AcousticWave* should have a virtual method *computeSamples(float duration, float frequency, float sampleRate)* that will be implemented in all its subclasses. In each of these implementations, the sample array from the class will be populated according to the shape of the wave and to the other properties (sample rate, frequency and amplitude). In the superclass, this method should just create an array of the appropriate size filled with zeros.

The formulas for each of the four wave types are presented below.

Sine wave

$$y_{si}(t) = A \cdot sin(2\pi f t)$$

Square wave

$$y_{sq}(t) = sign(y_{si}(t))$$

Triangle wave

$$y_{tr}(t) = \frac{2A}{\pi} \cdot arcsin(sin(2\pi f t))$$

Saw tooth wave

$$y_{st}(t) = -\frac{2A}{\pi} \cdot arctan(cot(t\pi f))$$

In the formulas above *A* stands for the amplitude of the wave, *f* is the frequency of the wave, *t* is the current timestamp.

Now, create a new class *WaveOutput* that contains a single static method that dumps the wave samples of an *AcousticWave* into a text file with the extension *.csv*.

> The signature of this method should be:
> *bool saveSamples(char* csv_path, AcousticWave *w);*

Each sample should be saved on a different row in this file. We can open these files with *Microsoft Excel* or *LibreOffice* and plot the values in a line chart. The obtained plot should be in accordance to the shape of the waves.

Finally, we will practice some operator overloading.
Overload the *operator<<* for all of the classes. Here we should display the type of the wave (*sine, square, triangle* or *sawtooth*), the frequency of the wave and the sound samples.

Finally, in the base class we will overload *operator **. This operator will be used to perform amplitude modulation on two sound samples: in this operation, "the amplitude of the carrier signal is varied in accordance with the instantaneous amplitude of the modulating signal". Although this operation sounds complicated, actually it just involves an element wise multiplication of two sound samples. This operation should be done in-place (i.e. it should also modify the sound sample of the first operand).

Generating the wav file

You can also generate an audio file using the script *save_to_wav.py,* and then use VLC to play the sound that you generated.

First, make sure you have Python 3 installed. Then open a terminal or command prompt and run:
`pip install -r requirements.txt`

Use the command below to convert the CSV into a WAV file:
`python save_to_wav.py --input sine_wave.csv --output sine_wave.wav`

The parameters of the script are :
--input: the input scv file (in this case sine_wave.csv, but replace it with the name of your CSV file)

--output: the output wav file (in this case sine_wave.wav, but replace it with your desired output filename)

--rate the sample rate (by default it is 44100 Hz).

When generating a wave, it's important to choose appropriate values for both frequency and sample rate. The frequency determines the pitch of the sound—how "high" or "low" it sounds. For example, **440 Hz frequency** is the standard musical note A4 and is a good choice for testing, as it's clearly audible and pleasant. The sample rate defines how many samples per second are used to digitally represent the wave. A typical sample rate value is **44,100 Hz (44.1 kHz)** which provides good sound quality. Always make sure the sample rate is at least twice the frequency to avoid distortion (a concept known as the Nyquist theorem).