Homework 5

This assignment is due **Sunday, March 4th, by 6 pm**

- All components (Cloud9 workspace, moodle quiz attempts, and zip file) must be completed and submitted by Sunday, March 4th, by 6:00 pm for your solution to receive points.

Please follow the same submission guidelines outlined in Homework 3 description regarding Style and Comments. You should not write test cases in main for this assignment.

- Develop in Cloud9: For this assignment, write and test your solution using Cloud9.
- Submission: All three steps must be fully completed by the submission deadline for your homework to receive points. Partial submissions will not be graded.

1. Make sure your Cloud 9 workspace is shared with your TA.
2. Your recitation TA will review your code by going to your Cloud9 workspace. TAs will check the last version that was saved before the submission deadline.
   - Create a directory called Hmwk5 and place all your file(s) for this assignment in this directory.
3. Submit to the Moodle Autograder: Head over to Moodle to the link *Hmwk5*. You will find one programming quiz question for each problem in the assignment. You can modify your code and re-submit (press Check again) as many times as you need to, up until the assignment due date.
4. Submit a zip file to Moodle: After you have completed all the questions and checked them on Moodle, you must submit a zip file with the .cpp file(s) you wrote in Cloud9. Submit this file going to *Hmwk 5 (File Submission)* on moodle.

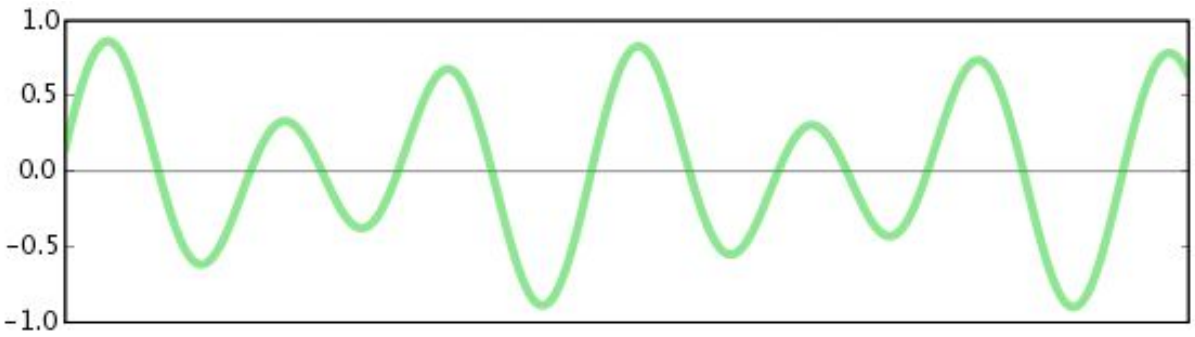
Background

This section is constructed of excerpts from the following source:

Audacity. “Digital Audio Fundamentals.” Audacity, manual.audacityteam.org/man/digital_audio.html.

Digital Sampling

All sounds we hear with our ears are *pressure waves* in air. Starting with Thomas Edison's demonstration of the first phonograph in 1877, it has been possible to capture these pressure waves onto a physical medium and then reproduce them later by regenerating the same pressure waves. Audio pressure waves, or waveforms, look something like this:

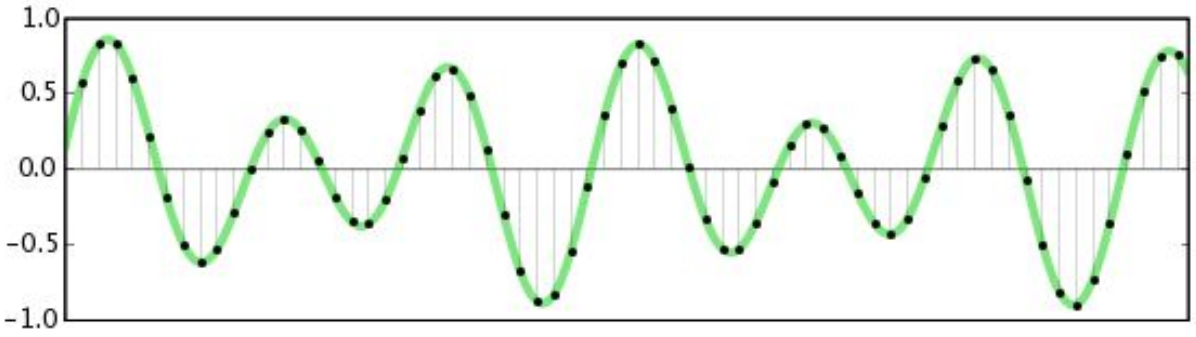


The x axis in this graph represents time, and the y axis represents the pressure.

Analog recording media such as a phonograph records and cassette tapes represent the shape of the waveform directly, using the depth of the groove for a record or the amount of magnetization for a tape. Analog recording can reproduce an impressive array of sounds, but it also suffers from problems of noise. Notably, each time an analog recording is copied, more noise is introduced, decreasing the fidelity. This noise can be minimized but not completely eliminated.

Digital recording works differently: it ***samples*** the waveform at evenly-spaced timepoints, representing each sample as a precise number. Digital recordings, whether stored on a compact disc (CD), digital audio tape (DAT), or on a personal computer, do not degrade over time and can be copied perfectly without introducing any additional noise. The following image illustrates a sampled audio

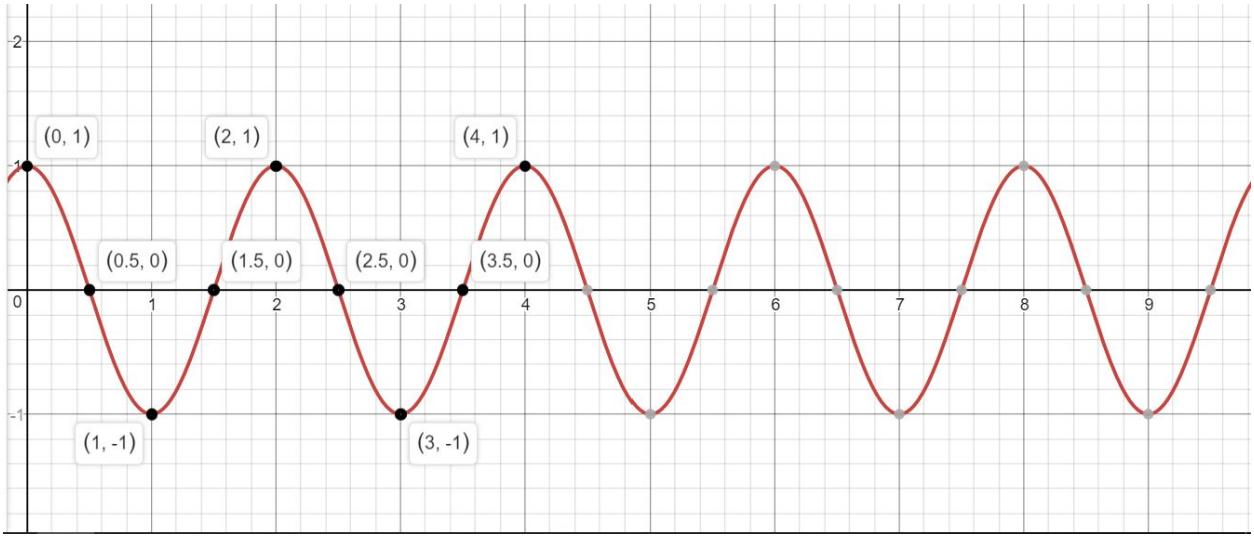
waveform. Each black dot represents a single **sample**. These samples can be used to reconstruct the wave when the recording is played back later.



Digital audio can be edited and mixed without introducing any additional noise. In addition, many digital effects can be applied to digitized audio recordings, for example, to simulate reverberation, enhance certain frequencies, or change the pitch.

Sample rates

Sample rates are measured in hertz (Hz), or cycles per second. This value is the number of samples captured per second.



If we set our **sampling rate = 2** (2 samples per second or a sample every ½ seconds) we can convert the waveform above into an array **samples** that looks like this:

```
[1, 0, -1, 0, 1, 0, -1, 0, …]
```

The value at `samples[0]` is the y value at time=0. The value at samples[1] is the value at `time = 0.5`. The value at `samples[2]` is the value at `time = 1.0`, and so on.

Higher sample rates allow higher audio frequencies to be represented. The human ear is sensitive to sound patterns with frequencies between approximately 20 Hz and 20000 Hz. Sounds outside that range are inaudible. Therefore a sample rate of 40000 Hz is the absolute minimum necessary to reproduce sounds within the range of human hearing.

Problem Set

For this assignment you will modify the **`sound.cpp`** we have provided on Moodle. There are also several .wav sound files.

Before you make any modifications to the code, verify that the existing **sound.cpp** works correctly. The existing program prompts the user for a sound file and makes that sound three times as loud.

- Download and play **cow.wav** on an audio player on your computer.
- Upload **sound.cpp** and **cow.wav** to the same directory in Cloud9.
- Make a copy of **cow.wav** in the same directory.
  - `cp cow.wav cow1.wav`
- Run **sound.cpp**.
  - When asked for a file name, provide **cow1.wav**.
- Download **cow1.wav** off of Cloud9 and play it to verify that the sound is three times as loud as the original **cow.wav**.

Be careful when making modifications to **sound.cpp**. We have provided comments specifying where you should be making changes. If you delete/change other portions of the code, your **sound.cpp** may not work correctly.

You should make a new copy of the sound file from the original every time you run **sound.cpp** (otherwise you will overwrite your original file).

The coderunner questions test only your functions, but you must submit a working **sound.cpp** file.

You are ***not required*** to write test cases in main for your function.

You are free to test using your own sound files. Make sure they are 16 bit mono WAV files. You should limit the length of your audio files to a few seconds so that the program does not take too long to run.

Question 1

Write a function to *reverse* a sound.

Reference the function `void process(int samples[], int size)` provided in `sound.cpp` to write your own function **`reverse.`** The two parameters in `process` function are array samples of int type and the size of this array. **reverse** should take the same parameters in the same order and modify the `samples[]` array so that the elements are in reverse order. The function will not return any values.

Examples

```
[0, 1, 2, 3]  --> [3, 2, 1, 0]

[3, 2, 1, 0]  --> [0, 1, 2, 3]

[5, 8, 3, 10] --> [10, 3, 8, 5]
```

Question 2

Write a function **`add_echo`** that adds an echo to the samples array. Your function should take four parameters: the samples array, the size of the samples array, the sampling rate (as an int), and the delay *d* in seconds (as a double), in this order. The function will not return any values.

When testing the audio files provided, a sampling rate of 40000 should be provided.

How to add an echo:

For each sample in the samples array, add the value from *d* seconds ago.

If there is not a sample at exactly *d* seconds ago, add the value from the sample at the closest sampling time greater than *d*.

Examples:

Let the sampling rate be 10 and the delay be 0.25 seconds. Since we have 10 samples a second, we have one sample every 1/10 seconds (0.1 seconds).

```
    //time (in          .0   .1   .2   .3   .4
     seconds)
int samples[] =      { 1,   4,   2,  5,    10   }
```

Let's say we are currently trying to add an echo with a delay d=0.25 seconds to just samples[4].

`samples[4]` is taken at `t=0.4`, and the desired delay is 0.25 seconds, so we want to get the value at:

```
time = 0.4 - 0.25  = 0.15 seconds
```

However, there is no sample at `t = 0.15`, so we add the value from the closest sampling time *after* `t = 0.15`, which is `t = 0.2` at `samples[2]` in this case.

```
//time =              .0   .1   .2   .3   .4
                    [ 1,   4,   2,  5,   12  ]
```

1. add_echo `([5, 8, 3, 10, 6, 2] , 6, 1, 1)`

Sampling rate is 1 (meaning 1 sample per second) and the delay is 1 second. For the echo, we need to add the value from 1 second before, which means the previous position.

`[5, 8, 3, 10, 6, 2] → [5, 8+5, 3+8, 10+3, 6+10, 2+6]`

`→ [5, 13, 11, 13, 16, 8]`

2. **add_echo** `([5, 8, 3, 10, 6, 2] , 6, 1, 2)`

Sampling rate is 1 (meaning 1 sample per second) and the delay is 2 seconds. For the echo, we need to add the value from the 2 seconds ago. Given the sampling rate of 1 sample per second, this means we need to add the value from two positions before

`([5, 8, 3, 10, 6, 2] , 6, 1, 2) → [5, 8, 3+5, 10+8, 6+3, 2+10]`

`→ [5, 8, 8, 18, 9, 12]`

3. **add_echo** `([5, 8, 3, 10, 6, 2] , 6, 2, 2)`

Sampling rate is 2 (meaning 2 samples per second) and the delay is 2 seconds. For the echo, we need to add the value from the 2 seconds ago. Given the sampling rate of 2 samples per second, this means we need to add the value from four positions before

`([5, 8, 3, 10, 6, 2] , 6, 2, 2) → [5, 8, 3, 10, 6+5, 2+8]`

`→ [5, 8, 3, 10, 11, 10]`

4. **add_echo** `([5, 8, 3, 10, 6, 2] , 6, 2, 1.5)`

Sampling rate is 2 (meaning 2 samples per second) and the delay is 1.5 seconds. For the echo, we need to add the value from the 1.5 seconds ago. Given the sampling rate of 2 samples per second, this means we need to add the value from three positions before.

`([5, 8, 3, 10, 6, 2] , 6, 2, 1.5) → [5, 8, 3, 10+5, 6+8, 2+3]`

`→ [5, 8, 3, 15, 14, 5]`

Note: the values of the highlighted samples will stay unchanged from their original value

Question 3

Write a function `normalize` to normalize a sound. Its parameters should be the same as those for `reverse()`.

Normalizing the sound samples array is a two step process.

- Find the maximum (highest) value in the samples array.
- Multiply all values in the samples array by 36773 and divide by the maximum value.

Examples

```
[0, 1, 2, 3]  --> [0, 1*36773/3, 2*36773/3, 3*36773/3]

[3, 2, 1, 0]  --> [3*36773/3, 2*36773/3, 1*36773/3, 0]

[5, 8, 10]    --> [5*36773/10, 8*36773/10, 10*36773/10]
```

Submitting Your Code to the Autograder on Moodle

You must name the functions as indicated in each problem description. The cout formats provided for each problem are not suggestions – they *MUST* be followed precisely, word-for-word and including all punctuation marks, otherwise the autograder will not recognize your results and you will not receive credit.

If there are errors in your solution to a particular problem, a button labeled “Show differences” will appear below the table of tests after you hit “check”. This can be a very useful tool in helping you find small typos, especially in cout statements.