Today's plan

- Review
- Midterm Project

Ex 6-2 - linked list

- `add_front(Node** list_ptr, char val);`
  - Crate new node with `data`.
  - Update new node's next pointer first using `*list_ptr`.
  - Update `*list_ptr` to point to the new node.
- `remove_after(Node* cur);`
  - Set a temp pointer to point to `cur->next`.
  - Update `cur->next` to `cur->next->next`.
  - Free the removed node using the temp pointer.
- `remove_front(Node** list_ptr);`
  - Set a temp pointer to point to `*list_ptr`.
  - Update `*list_ptr` to `(*list_ptr)->next`.
  - Free the removed node using the temp pointer.
- `remove_all(Node** list_ptr, char val);`
  - Iteration: call `remove_front` when the front node matches. Then call `remove_after` when the next node matches.
  - Recursion: if the front node matches call `remove_front`, else update `list_ptr` to point to the next node. Recursive calls using `list_ptr`.

Ex 6-2 - linked list

- `insert(Node** list_ptr, char val);`
  - Iteration: iterate the list until the `data` is grater than or equal to the input `val`, then call `add_front`.
  - Recursion: if current node's `data` is greater than or equal to the input `val`, call `add_front`, else recursive call using `&(*list_ptr)->next`.
- other functions using similar logic. Try to finish them at home and practice recursions.
  - `add_tail`
  - `find`
  - `remove_char`
  - `replace`

Midterm Project

- Audio processing
- Detail description: `https://jhu-ip.github.io/cs220-sp21/docs/assignments/midterm`
- Starter codes: `https://github.com/jhu-ip/cs220-sp21-public/tree/master/midterm`
    - Complete `io.c`
    - Complete `wave.c`
    - Write three programs: `render_tone.c`, `render_song.c`, and `render_echo.c`
    - Write a `Makefile` that builds the three programs
    - Write a `README` and provide the `gitlog`
- **Carefully read all the instructions given in the description.**

Midterm Project - Digital sound

How do we model sounds?

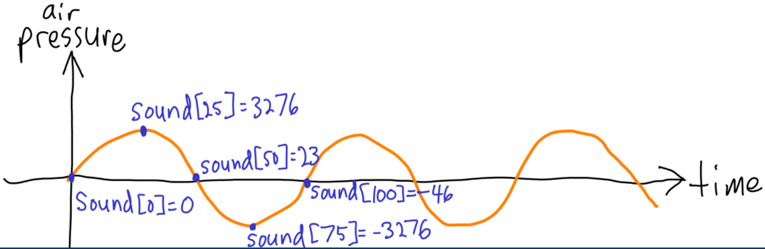
- Model by different period waveforms, e.g. sine waves, square waves, and sawtooth waves, which are the three waveforms you need to generate in this project.
- Waveforms are periodic, so they repeat themselves in time. They also have a magnitude, which we call the ‘air pressure’ or simply the intensity of the sound.
- Sound waves interact additively (simply adding their intensity) to form a complex sound.

How do we represent sound digitally? (WAVE files)

- Sample the sound at a particular frequency (e.g. 44.1 kHz)
- Store the samples as an array of `int16_t` (an array of 16 bits signed integers)
- For stereo sounds, even indexed samples belong to the left channel while odd belong to the right.

Midterm Project - Digital sound

- How do we sample sounds to an array?
    - Given a particular frequency, we know at which time we take a sample.
    - Knowing how to generate different waveforms, we can compute the intensity at a specific time.
    - As we model sounds as different waveforms added together, we can sample each waveform and sum up the intensities.
    - The resulting intensity can be stored them into an `int16_t` array.
- For example:



Midterm Project - Sampling and waveforms and ‘gain’

- Sampled values are represented by an `int16_t` array
  - it has a range between -32,768 and 32,767.
  - when it is underflow/overflow during additions, you need to **clamp** the values.
  - for stereo sounds, remember to use even indexed samples for the left channel and odd for the right.
  - the stereo array will be written in a binary file (WAV files). (beware of the endianness, if you have a big endian machine.)
- Three waveforms to generate:
  - sine waves (`generate_sine_wave`): use the formula provided.
  - square waves (`generate_square_wave`): use sine waves - set positive values to maximum and negative values to minimum (w.r.t. the amplitude).
  - sawtooth waves (`generate_saw_wave`): linearly increasing from minimum to maximum (w.r.t. the amplitude) in a cycle.
- ‘gain’ - a factor to rescale the sampled values
- `apply_gain` function: apply a ‘gain’ to all the sampled values.

Midterm Project - ADSR envelop

- ADSR (attack, decay, sustain, release) envelope - `apply_adsr_envelope`
    - modify the sounds by recalling the samples with different 'gains'
    - 'gains' are different, depending on which phase the sample falls in the envelope.
    - if there are enough samples to cover the envelope
        - attack phase: 'gain' increases linearly from 0 to 1.2.
        - decay phase: 'gain' decreases linearly from 1.2 to 1.0.
        - sustain phase: 'gain' is 1.0, i.e. samples are unchanged.
        - release phase: 'gain' decreases linearly from 1.0 to 0.
    - else it only has two phases - rise and fall
        - rise phase: 'gain' increases from 0 to 1.0.
        - fall phase: 'gain' decreases from 1.0 to 0.
    - In a word, you need to check which phase a sample falls into and compute the corresponding 'gain' to rescale the sample.
    - Clamping may be needed if it is out of the 16 bits signed integer range.

Midterm Project - Mix in and panning

- `mix_in` is a function to 'add' mono sampled data to the stereo data, which will be written to WAVE files.
- Clamping may be needed if it is out of the 16 bits signed integer range.
- `compute_pan`: a function to compute gains for each channel using a formula.
- All the above mentioned functions are recommended but not necessary.

Midterm Project - Important requirements

- `fetal_error` function.
- Use `struct` to represent 'instrument'.
- The three render programs:
  - `render_tone`
  - `render_song`
  - `render_echo`
- Always follow the submission requirements
- Highly recommend to test all your helper functions before you start implementing the three programs

Midterm Project - The three programs

- `render_tone`
  - Input: the waveform type (0,1,2), frequency (determine how long is 1-cycle), amplitude (determine the min and max intensity of the samples), numsamples (usually 44,100), wavfileout (output filename).
  - Output: a WAV file of the specified waveform (sine, square, sawtooth).
- `render_song`
  - Input: songinput (input song text filename), waveoutput (output filename).
  - Output: a WAVE file that plays the MIDI notes specified in the song text file.
  - Note: there are at most 16 instruments. i.e. at most 16 mono sampled values.
  - You will use `mix_in` to mix them into the stereo data.
  - Keep in mind that left and right channels may have different gains.

Midterm Project - The three programs

- `render_echo`
  - Input: wavfilein (input filename), wavfileout (output filename), delay (number of samples to delay), amplitude (of the echo effect - a percentage of original magnitude).
  - Output: a WAV file that adds the echo effect to the input WAV file.
  - Read in the input to an array and extend to include the delayed samples.
  - Create new array for the echo effect.
  - Copy from original array to the new buffer by shifting it for the delay.
  - Rescale the echo array by `amplitude` and mix in.