```
'''
import mido



# load the MIDI file
midi_file = mido.MidiFile('eyetrackingoutput.mid')

note_events = []  # to store note events WITH timing

# iterate through MIDI messages
for track in midi_file.tracks:
   current_time = 0
   for msg in track:
       current_time += msg.time  # Accumulate the time for each
message
       if msg.type == 'note_on' or msg.type == 'note_off':
           # append relevant note info and timing to note_events
           note_events.append({
               'type': msg.type,
               'note': msg.note,
               'velocity': msg.velocity,
               'time': current_time
           })

print(note_events) # for debugging



ticks_per_beat = midi_file.ticks_per_beat
bpm = 120
ms_per_tick = (60000 / (bpm * ticks_per_beat))

for event in note_events:
```

```
    event['time_ms'] = event['time'] * ms_per_tick

## need to send to appropriate ports?

with open('note_data.h', 'w') as f:
    f.write("NoteEvent note_schedule[] = {\n")
    for event in note_events:
        time_ms = int(event['time_ms'])
        note = event['note']
        velocity = event['velocity']
        f.write(f"    {{ {time_ms}, {note}, {velocity} }},\n")
    f.write("};\n")

'''

# hi, this code below produces some unwanted phantom notes, but no weird zero-duration notes
import mido
import time

def parse_midi_file(midi_file_path):
    """
    Parses a MIDI file and extracts note-on events with their respective durations.

    Params:
        midi_file_path (str): Path to the MIDI file

    Returns:
        list of tuples, wherein each tuple contains the note and its duration in seconds
    """
```

```
    midi = mido.MidiFile(midi_file_path)
    notes = []
    active_notes = {}  # dictionary to store start times for
active notes
    absolute_time = 0  # tracks total elapsed time from start of
the track

    for msg in midi:
        absolute_time += msg.time

        # check for a note-on event with velocity > 0 (start of a
note)
        if msg.type == 'note_on' and msg.velocity > 0:
            active_notes[msg.note] = absolute_time  # record the
start time of the note

        #cCheck for note-off events OR note-on with velocity 0
(end of a note)
        elif msg.type in ['note_off', 'note_on'] and msg.note in
active_notes:
            # calculate duration based on the difference from
start time
            start_time = active_notes.pop(msg.note)
            duration = absolute_time - start_time
            notes.append((msg.note, duration))  # append note
WITH its duration

    return notes

def simulate_note_playback(notes):
    """
```

```
    Simulates playback of notes by printing each note and waiting
for its duration.

    Parameters:
        notes (list of tuples): Each tuple contains a note and
its duration in seconds.
    """
    print("Starting simulated note playback...")
    for note, duration in notes:
        print(f"Playing note: {note} for {duration:.2f} seconds")
        time.sleep(duration)
    print("Playback complete.")

def format_notes_for_firmware(notes):
    """
    Formats notes into a structure suitable for firmware
transmission.

    Parameters:
        notes (list of tuples): Each tuple contains a note and
its duration.

    Returns:
        str: Formatted data string ready for UART transmission.
    """
    formatted_data = ""
    for note, duration in notes:
        formatted_data += f"Note: {note}, Duration:
{duration:.2f}s\n"
    return formatted_data

if __name__ == "__main__":
```

```
    midi_file_path = "/Users/shravyaks/Documents/sample.mid"
    notes = parse_midi_file(midi_file_path)
    print("Parsed notes:", notes)

    simulate_note_playback(notes)

    # Format data as it would be for firmware transmission
    firmware_data = format_notes_for_firmware(notes)
    print("\nFormatted data for firmware transmission:\n",
firmware_data)
```