Useful Constants

### Midi Program Numbers

#### Ensemble

- 49 **String Ensemble 1**
- 50 **String Ensemble 2**
- 51 **Synth Strings 1**
- 52 **Synth Strings 2**
- 53 **Choir Aahs**
- 54 **Voice Oohs**
- 55 **Synth Choir**
- 56 **Orchestra Hit**

#### Brass

- 57 **Trumpet**
- 58 **Trombone**
- 59 **Tuba**
- 60 **Muted Trumpet**
- 61 **French Horn**
- 62 **Brass Section**
- 63 **Synth Brass 1**
- 64 **Synth Brass 2**

#### Reed

- 65 **Soprano Sax**
- 66 **Alto Sax**
- 67 **Tenor Sax**
- 68 **Baritone Sax**
- 69 **Oboe**
- 70 **English Horn**
- 71 **Bassoon**
- 72 **Clarinet**
<table>
<thead>
<tr>
<th>Channel Voice Messages [nnn = 0-15 (MIDI Channel Number 1-16)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000nnn 0kkkkkk 0vvvvvv Note Off event.</td>
</tr>
<tr>
<td>This message is sent when a note is released (ended).</td>
</tr>
<tr>
<td>(kkkkkk) is the key (note) number. (vvvvvv) is the</td>
</tr>
<tr>
<td>velocity.</td>
</tr>
<tr>
<td>1001nnn 0kkkkkk 0vwwww Note On event.</td>
</tr>
<tr>
<td>This message is sent when a note is depressed (start).</td>
</tr>
<tr>
<td>(kkkkkk) is the key (note) number. (vwwwww) is the</td>
</tr>
<tr>
<td>velocity.</td>
</tr>
<tr>
<td>1010nnn 0kkkkkk 0vwwww Polyphonic Key Pressure (Aftertouch).</td>
</tr>
<tr>
<td>This message is most often sent by pressing down on the key</td>
</tr>
<tr>
<td>after it &quot;bottoms out&quot;. (kkkkkk) is the key (note) number.</td>
</tr>
<tr>
<td>(vwwwww) is the pressure value.</td>
</tr>
<tr>
<td>1011nnn 0cccccccc 0vwwww Control Change.</td>
</tr>
<tr>
<td>This message is sent when a controller value changes.</td>
</tr>
<tr>
<td>Controllers include devices such as pedals and levers.</td>
</tr>
<tr>
<td>Controller numbers 120-127 are reserved as &quot;Channel Mode</td>
</tr>
<tr>
<td>Messages&quot; (below), (cccccccc) is the controller number (0-119)</td>
</tr>
<tr>
<td>(vwwwww) is the controller value (0-127).</td>
</tr>
<tr>
<td>1100nnn 0ppppppp Program Change. This message sent when the</td>
</tr>
<tr>
<td>patch number changes. (pppppp) is the new program number.</td>
</tr>
<tr>
<td>1101nnn 0vwwwww Channel Pressure (After-touch). This message</td>
</tr>
<tr>
<td>is most often sent by pressing down on the key after it</td>
</tr>
<tr>
<td>&quot;bottoms out&quot;. This message is different from polyphonic</td>
</tr>
<tr>
<td>after-touch. Use this message to send the single greatest</td>
</tr>
<tr>
<td>pressure value (of all the current depressed keys).</td>
</tr>
<tr>
<td>(vwwwww) is the pressure value.</td>
</tr>
<tr>
<td>1110nnn 0lllllll 0mmmmmmmm Pitch Wheel Change. Ommmmmmmm This</td>
</tr>
<tr>
<td>message is sent to indicate a change in the pitch wheel.</td>
</tr>
<tr>
<td>The pitch wheel is measured by a fourteen bit value.</td>
</tr>
<tr>
<td>Center (no pitch change) is 20480H. Sensitivity is a function</td>
</tr>
<tr>
<td>of the transmitter. (lllllll) are the least significant 7 bits,</td>
</tr>
<tr>
<td>(mmmmmmmm) are the most significant 7 bits.</td>
</tr>
</tbody>
</table>

Table 3: Control Changes and Mode Changes (Status Bytes 176-179)

<table>
<thead>
<tr>
<th>Control Number (2nd Byte Value)</th>
<th>Control Function</th>
<th>3rd Byte Value</th>
<th>Value</th>
<th>Used As</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000 06</td>
<td>Bank Select</td>
<td>0-127</td>
<td>MSB</td>
<td></td>
</tr>
<tr>
<td>00000001 01</td>
<td>Modulation Wheel or Lever</td>
<td>0-127</td>
<td>MSB</td>
<td></td>
</tr>
<tr>
<td>00000010 02</td>
<td>Breath Controller</td>
<td>0-127</td>
<td>MSB</td>
<td></td>
</tr>
<tr>
<td>00000011 03</td>
<td>Undefined</td>
<td>0-127</td>
<td>MSB</td>
<td></td>
</tr>
<tr>
<td>00000100 04</td>
<td>Foot Controller</td>
<td>0-127</td>
<td>MSB</td>
<td></td>
</tr>
<tr>
<td>00000101 05</td>
<td>Portamento Time</td>
<td>0-127</td>
<td>MSB</td>
<td></td>
</tr>
<tr>
<td>00000110 06</td>
<td>Data Entry MSB</td>
<td>0-127</td>
<td>MSB</td>
<td></td>
</tr>
<tr>
<td>00001111 07</td>
<td>Channel Volume (formerly Main Volume)</td>
<td>0-127</td>
<td>MSB</td>
<td></td>
</tr>
<tr>
<td>00010000 08</td>
<td>Balance</td>
<td>0-127</td>
<td>MSB</td>
<td></td>
</tr>
<tr>
<td>00010001 09</td>
<td>Undefined</td>
<td>0-127</td>
<td>MSB</td>
<td></td>
</tr>
<tr>
<td>00010100 0A</td>
<td>Pan</td>
<td>0-127</td>
<td>MSB</td>
<td></td>
</tr>
<tr>
<td>00010101 0B</td>
<td>Expression Controller</td>
<td>0-127</td>
<td>MSB</td>
<td></td>
</tr>
<tr>
<td>00011000 0C</td>
<td>Effect Control 1</td>
<td>0-127</td>
<td>MSB</td>
<td></td>
</tr>
<tr>
<td>00011010 0D</td>
<td>Effect Control 2</td>
<td>0-127</td>
<td>MSB</td>
<td></td>
</tr>
</tbody>
</table>

CMSIP Spring 2019 PRACTICE Final Exam
1) MIDI

a) Reza Vali wanted to explore Persian tuning using off-the-shelf MIDI synthesizers. MIDI assumes an equal-tempered tuning where chromatic scale steps are equally spaced (12 to the octave), whereas Persian tuning is based on a set (scale) of pitches that are not in our Western chromatic scale. What MIDI message is available to slightly adjust the pitch of a MIDI note?

b) In Reza’s strategy, there’s a problem with fine-tuning the pitch of each MIDI note. How many notes can you play simultaneously with fine-grain pitch control given a single conventional MIDI connection and synthesizer? (This is a problem related to MIDI encoding, naming notes, and scope of control messages.)

c) A related problem is controlling the instrument of each note (e.g. piano, flute, bell, or bass sound). What message is used to select the instrument for a note?

d) How many different instruments can you play simultaneously given a single conventional MIDI connection and synthesizer? (Ignore the special case of drum sounds.)

e) About how long (in milliseconds) does it take to send 10 note-on messages in MIDI? (Use $B = \text{bits-per-second}$ as a parameter if you do not know it.)

f) How many bytes does it take to send 10 note-on messages in MIDI?
2) **Audio Processing**

a) Label the following graph of unit generators with numbers to indicate the order in which they should be computed. An arrow from A to B indicates that data generated by A is read by B.

```
Audio Input → abs(x) → LPF → Sine → delay → * gain1 → + → Audio Output
```

b) Some systems like STK traverse a graph as shown in (a), and for each unit generator, a method is called that returns one sample of audio. Traversing the graph and calling all those methods is a lot of work just to compute one sample. How do most unit generator systems improve efficiency?
c) Application programs are often scheduled every 1ms, but operating systems may schedule programs only every 50ms or so. As some low level in the hardware, each sample must be delivered at the sampling rate, which is often 44,100Hz, or one sample every 22µs. How do systems deal with the 50:1 or even 2500:1 disparity between scheduling periods (~1ms or ~50ms) and sample periods (~20µs)?

3) **Design question.** We heard from Roger Linn about the importance of low jitter in musical instrument response. Consider a system that takes input via OSC (or O2) from digital instrument sensors and uses software to synthesize audio output in response to data from the instrument. The synthesis software runs PortAudio, and you establish that PortAudio callbacks have a jitter of up to 25ms (i.e. you might expect a callback every 2ms, but in fact, sometimes there is a pause of 23ms followed by 10 or so instant callbacks to "catch up." Thus, if the callbacks act on the latest sensor data, this 25ms of jitter will be passed on to the output sound.

![Diagram of the system](image)

a) Design an implementation that reduces jitter to (at most) the nominal period of PortAudio callbacks (e.g. 2ms in this example). You do not need to write code, but explain the key elements and concepts of the design. It should be clear that your design will reduce jitter.
b) Be sure to consider additional sources of jitter. If not already addressed in part (a), what assumptions do you make about (i) the instrument itself, (ii) the network connection for OSC or O2, (iii) the behavior of the “Synthesis App” application receiving the messages? (note that it would be unreasonable to assume the app has very low jitter and ideal scheduling given that the high-priority PortAudio thread has 25ms of jitter).
c) Typically, when you reduce jitter, you must increase latency. In your design, and based on your assumptions, give a reasonable estimate of the total latency from instrument “gesture” (physical input) to sound output.

1) **Music Generation and Algorithmic Composition** (4 pts)
   a) Name a limitation of Markov Chains for music generation.

   b) Other than Markov Chains, name two other approaches to Algorithmic Composition that we discussed in class.
4) **Theory & Music Representation**

![Musical Score Image]

a) What is the key signature?

b) What is the time signature?

c) How many complete measures are notated in the score?

d) How many parts are in this score?

e) How long does it take to perform a measure by strictly following the score’s tempo?

f) Translate the first full measure of the score (ignore the first or “pickup” note in the violin), and consider the 2nd violin note to be at time 0) into a Serpent note list representation: Use an array of notes where each note is represented by an array of the form

\[
\text{[time, duration, voice, pitch, velocity]},
\]

where

i) time and duration are in seconds (floating point),

ii) voice is 'violin' or 'cello' (recall Serpent symbols are denoted by single quotes)

iii) pitch is a MIDI key number (integer), and

iv) velocity is a MIDI velocity number (integer, pick a reasonable value).
2) Networks and Music Control (6 pts – 2 each)
   a) O2 has two forms of send. `o2_send()` sends a message as quickly as possible but does not ensure delivery. `o2_send_cmd()` sends a message reliably (the message is guaranteed to arrive unless the sender or receiver crashes first), but the message delivery may not be as fast. Write “TCP” or “UDP” to indicate how O2 delivers messages for:
      5) `o2_send()` __________
         `o2_send_cmd()` __________
   b) OSC generally uses (circle one): TCP or UDP ?

   c) Suppose an O2 service named “synth” offers two voices named “voice1” and “voice2” that each allow you to set “freq” (frequency) and “amp” (amplitude). Write an O2 address that could be used to set the frequency of voice1 to 440 Hz.

3) Standard MIDI Files
   a) Name two things that Standard Midi Files can represent that are not (easily) representable in Common Practice Music Notation.

   b) Name two things that Common Practice Music Notation can represent that are not (easily) representable in Standard Midi Files.

6) Clock Synchronization (6 pts – 3 each)
   Fill in the blanks. The clock synchronization algorithm described in class estimates the difference between a slave’s local clock and the clock of the master. The protocol is basically:

   ![Clock Synchronization Diagram](image)
1. Run steps (a) through (e) for \( i = 0, 1, \ldots, N-1 \):
   a. at slave time \( S_i \), the slave sends a message to the master
   b. at master time \( M_i \), the master replies with \( M_i \)
   c. at slave time \( T_i \), the slave receives \( M_i \)
   d. the slave estimates that the local time corresponding to master time \( M_i \) is
      \[
      L_i = \text{__________________________}.
      \]
   e. The local clock offset is estimated as \( \Delta_i = L_i - M_i \).
2. Finally, the algorithm computes an overall estimated \( \Delta \) as follows: (insert short description)
7) Music Understanding and Classifiers

Recall that \( P(A \& B) = P(A|B)P(B) = P(B|A)P(A) \).

Suppose you want to classify whether a piece of music is Happy or Sad, and there's only one feature, the mean pitch. Your job is to guess the most likely music style given the observed feature using a Gaussian naïve Bayes classifier. Observed statistics of the mean and standard deviation for Happy and Sad music is shown in the table below. The prior guess for the two music styles are equal, i.e., \( P(\text{Happy}) = P(\text{Sad}) = \frac{1}{2} \). Recall that with a Gaussian distribution, the probability of observing a feature is decreasing with the number of standard deviations from the mean. In other words

\[
\frac{\left| \text{observation} - \mu_{\text{classA}} \right|}{\sigma_{\text{classA}}} < \frac{\left| \text{observation} - \mu_{\text{classB}} \right|}{\sigma_{\text{classB}}}
\]

if and only if

\[
p(\text{observation} \mid \text{classA}) > p(\text{observation} \mid \text{classB}).
\]

<table>
<thead>
<tr>
<th></th>
<th>Mean within class ( \mu_{\text{class}} )</th>
<th>Standard Deviation within class ( \sigma_{\text{class}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Happy</td>
<td>65</td>
<td>4</td>
</tr>
<tr>
<td>Sad</td>
<td>57</td>
<td>8</td>
</tr>
</tbody>
</table>

a) If the observed mean pitch is 61 (half-way between \( \mu_{\text{happy}} \) and \( \mu_{\text{sad}} \)), what is your guess for the music style? Show your work using the formulas given above.
b) Starting from the description in the previous problem, we change the prior probabilities for the styles to be not equal, letting $P(\text{Happy}) = 1/4$ and $P(\text{Sad}) = 3/4$. Redo the work under this assumption. What would be your guess for the music style?

c) What are 2 features you might use to classify styles or genre in MIDI files?
8) Concurrency

Consider the following code (in Serpent) that inserts a value into a linked list:\(^1\)

```
1 values = nil # values is a global variable, a list
2
class Node:
3    var value
4    var next
5
6 def insert(value):
7    # Insert value at front of values.
8    var node = Node()
9    node.value = value
10   node.next = values
11   values = node
12```

Serpent does not have preemptive threads, so this code is “safe” in Serpent, but for this problem, imagine a Serpent interpreter that can switch threads after any statement.

a) Give an example interleaving of two threads using the above insert() code (without locks) that results in losing data. You can assume each thread executes one call to insert and just write down the order of lines 9, 10, 11 and 12.

<table>
<thead>
<tr>
<th>Thread 1: insert(1)</th>
<th>Thread 2: insert(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

b) If the initial list contained just one Node with the value 3, what would the list contain after the sequence of statements you wrote in part (a)?

---

\(^1\) This is really terrible code: insert should be a method, values shouldn't be a global, insert should return a new list, but that all obscures the exam problem.
c) Add calls to `lock(k)` and `unlock(k)` to obtain correct behavior when `insert()` is called from multiple concurrent threads:

```javascript
var k = lock_create(); // create a new global lock object
void insert(int amt)
    # Insert value at front of values.
    var node = Node()
    node.value = value
    node.next = values
    values = node
```

d) We’ve learned about scheduling and events to obtain most of the advantages of concurrency without actually using multiple threads. When is this approach inadequate, motivating threads and preemptive scheduling? (Choose the best answer):

i) When you have input from MIDI or OSC
ii) When you have 100’s of activities.
iii) When some actions must be completed faster than the execution time of some other action.
iv) All of the above

e) One of the main uses of threads in music applications is to allow time-critical audio computation to preempt other code. (Otherwise, running audio processing later might cause the audio output buffer to empty, which would insert pops or gaps into the audio output). Allowing preemptive audio threads begs for locks to solve concurrency problems, yet most audio systems tell developers not to use locks at all in audio callbacks. Why?
9) Matching

Match each item at left with a corresponding item at right.

A. Chromagram  ___ hashing
B. Forward Synchronous Scheduling  ___ logical (or virtual) time
C. Genre Classification  ___ MAX
D. Music Fingerprinting  ___ Naïve Bayes
E. OSC  ___ time deltas
F. Standard MIDI File  ___ topological sort
G. Unit Generators  ___ UDP
H. Visual Programming  ___ vectors of length 12

10) Event-based, real-time programming

a) Write Serpent code to generate two sequences of MIDI notes. One sequence should play pitch F6 100 times with an initial inter-note onset (IOI) time of 2s, decreasing by 0.02s on each successive note. (Thus, the last IOI will be 0.02s.) The other sequence should play pitch G3 100 times with an initial IOI of 0.02s, increasing by 0.02s for each successive note. (Thus, the last IOI will be 2s, and the sequences will have the same duration.)

To play a note (don’t worry about the duration / note off), call (but do not define):
\texttt{note(pitch)}

To schedule a function call, call (but do not define):
\texttt{sched\_cause(time\_from\_now, nil, function\_name, p1, p2, ...)}

You can assume the scheduler in \texttt{sched.srp} is loaded and running in the normal way.

\textit{Important:} follow the template below. Define one function to play both sequences as shown.
def start_sequence(): // this is called to start both sequences


def play_next(pitch, ioi, ioi_inc, n): // a helper function
// pitch – pitch to play
// ioi – ioi to the next note,
// ioi_inc – change from this ioi to the next,
// n – number of (remaining) notes to play


// use only as many lines as you need
b) Modify your code (from the previous page) so that the sequence can be stopped. You may use a global variable. The function stop_sequence() will cause the sequence to stop. Your solution should work even if a start_sequence is called immediately after stop_sequence (in which case the old sequence is stopped before it plays another note, but a new sequence is started immediately).

// declare any global variables here:


def start_sequence(): // this is called to start both sequences


def stop_sequence(): // calling this halts any currently playing sequence


def play_next(pitch, ioi, ioi_inc, n, ____________):
    // pitch – pitch to play
    // ioi – ioi to the next note,
    // ioi_inc – change from this ioi to the next,
    // n – number of (remaining) notes to play


// use only as many lines as you need
11) Scheduling

a) The following graph shows a mapping from time in seconds to beats. Draw the second graph, which shows tempo as a function of time.

b) Write the mathematical equation for $T(b)$ (where $T(b)$ is time of beat $b$) in terms of $B(t)$ (where $B(t)$ is the beat position at time $t$).

c) Write the mathematical equation for $T(b)$, the time of beat $b$, in terms of $S(b)$, the tempo (in beats/second) at beat $b$. 
12) Audio Editors (6 pts – 2 each)

a) In Audacity, if you filter a signal and zoom in, you might see a change in the waveform. However, in ProTools, if you apply a filter to a signal, you do not see the waveform change on the display. Why?

b) Aside from display differences, name one advantage of in-place editors and Audacity over non-destructive editors.

c) Aside from display differences, name one advantage of non-destructive editors over in-place editors and Audacity.