15-323/623 Computer Music Systems and Information Processing Exam
Spring 2017

Note: 3 hour exam, 100 points total.

Midi Program Numbers

Useful Constants

| Midi number | Note name | Keyboard | Frequency | Period
|-------------|-----------|----------|-----------|--------
| 21-35       |           |          |           |        |
| 36-45       |           |          |           |        |
| 46-49       |           |          |           |        |
| 50-52       |           |          |           |        |
| 53-55       |           |          |           |        |
| 56-60       |           |          |           |        |
| 61-64       |           |          |           |        |

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
Channel Voice Messages \([nnn = 0-15 \text{ (MIDI Channel Number 1-16)}]\)

<table>
<thead>
<tr>
<th>Channel</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000nnn</td>
<td>000000</td>
<td>Note Off event. This message is sent when a note is released (ended). ((kkkkkkk)) is the key (note) number. ((vvvvvvv)) is the velocity.</td>
</tr>
<tr>
<td>1001nnn</td>
<td>000000</td>
<td>Note On event. This message is sent when a note is depressed (start). ((kkkkkkk)) is the key (note) number. ((vvvvvvv)) is the velocity.</td>
</tr>
<tr>
<td>1010nnn</td>
<td>000000</td>
<td>Polyphonic Key Pressure (Aftertouch). This message is most often sent by pressing down on the key after it &quot;bottoms out&quot;. ((kkkkkkk)) is the key (note) number. ((vvvvvvv)) is the pressure value.</td>
</tr>
<tr>
<td>1011nnn</td>
<td>000000</td>
<td>Control Change. This message is sent when a controller value changes. Controllers include devices such as pedals and levers. Controller numbers 120-127 are reserved as &quot;Channel Mode Messages&quot; (below). ((cccccccc)) is the controller number ((0-119)). ((vvvvvvv)) is the controller value ((0-127)).</td>
</tr>
<tr>
<td>1100nnn</td>
<td>000000</td>
<td>Program Change. This message sent when the patch number changes. ((ppppppp)) is the new program number.</td>
</tr>
<tr>
<td>1101nnn</td>
<td>000000</td>
<td>Channel Pressure (After-touch). This message is most often sent by pressing down on the key after it &quot;bottoms out&quot;. This message is different from polyphonic aftertouch. Use this message to send the single greatest pressure value of all the current depressed keys. ((vvvvvvv)) is the pressure value.</td>
</tr>
<tr>
<td>1110nnn</td>
<td>000000</td>
<td>Pitch Wheel Change. ((mmmmmmm)) This message is sent to indicate a change in the pitch wheel. The pitch wheel is measured by a fourteen bit value. Center (no pitch change) is 2000H. Sensitivity is a function of the transmitter. ((lllllll)) are the least significant 7 bits. ((mmmmmmm)) are the most significant 7 bits.</td>
</tr>
</tbody>
</table>

Table 3: Control Changes and Mode Changes
(Status Bytes 176-191)

<table>
<thead>
<tr>
<th>Control Number (2nd Byte Value)</th>
<th>Control Function</th>
<th>3rd Byte Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>Binary</td>
<td>Hex</td>
</tr>
<tr>
<td>0</td>
<td>00000000</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>00000001</td>
<td>01</td>
</tr>
<tr>
<td>2</td>
<td>00000010</td>
<td>02</td>
</tr>
<tr>
<td>3</td>
<td>00000011</td>
<td>03</td>
</tr>
<tr>
<td>4</td>
<td>00000100</td>
<td>04</td>
</tr>
<tr>
<td>5</td>
<td>00000101</td>
<td>05</td>
</tr>
<tr>
<td>6</td>
<td>00000110</td>
<td>06</td>
</tr>
<tr>
<td>7</td>
<td>00000111</td>
<td>07</td>
</tr>
<tr>
<td>8</td>
<td>00001000</td>
<td>08</td>
</tr>
<tr>
<td>9</td>
<td>00001001</td>
<td>09</td>
</tr>
<tr>
<td>10</td>
<td>00001010</td>
<td>0A</td>
</tr>
<tr>
<td>11</td>
<td>00001011</td>
<td>0B</td>
</tr>
<tr>
<td>12</td>
<td>00001100</td>
<td>0C</td>
</tr>
<tr>
<td>13</td>
<td>00001101</td>
<td>0D</td>
</tr>
</tbody>
</table>
1) **MIDI** (6pts – 1 each)

   a) Here is a three-byte MIDI message (in **hexadecimal**, if reading hexadecimal is a challenge you might want to complete columns 2 and 3 of Table 3 on page 2 for reference):

   \[97 \ 4C \ 73\]

   i) What type of message is this: (note-off, note-on, poly key pressure, control change, program change, channel pressure, pitch wheel change, etc.)?

   ii) Rewrite the 3-byte message in **hexadecimal**, setting everything to zero except the channel number:

   iii) Synthesizers, keyboards, and MIDI software usually display channel numbers in the range 1-16. What number (from 1-16) would you need in order to generate this message?

   iv) What does the 2\textsuperscript{nd} byte represent? And what is the maximum value it can have? (In **decimal and hexadecimal**)

   v) What does the 3\textsuperscript{rd} byte represent? And what is the maximum value it can have? (In **decimal and hexadecimal**)

   b) Pitch wheel change messages are used to “bend” pitches smoothly up or down by updating the pitch when a pitch wheel is moved by hand. To avoid a “stairstep” effect in which each pitch change is audible, the pitch changes should be very frequent. If nothing else is going on, about how many pitch wheel change messages can be sent over a standard MIDI connection per second. (Round numbers within 10\% are fine. If not sure of MIDI baud rate, you can use it as a variable. By the way, MIDI over USB is much faster.)
2) **Audio Processing** (8pts – 2, 2, 1, 1, 2)
   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 audio flows from A to B.

   ![Graph of unit generators]

   b) Label this timing diagram of a double-buffered audio output scheme. The top row depicts timing of adding audio samples to buffers, while the bottom row indicates timing of moving samples from buffers to the digital-to-analog converter (DAC). Label the diagram with
   o W1 (period(s) during which samples are written to buffer 1),
   o W2 (period(s) during which samples are written to buffer 2),
   o R1 (period(s) during which samples are read from buffer 1 and copied to the DAC), and
   o R2 (period(s) during which samples are read from buffer 2 and copied to the DAC).

   ![Timing diagram of double-buffered audio output scheme]
c) For at least one of the periods marked W1 (writing buffer 1), mark with B when filling the buffer can begin, and with D when is the deadline for completing the computation and sample writing operation.

d) Somewhere in the picture, draw a double arrow line, e.g. ←----L----→ labeled with L to show the latency from when you start computing the first sample of a buffer to when that first sample is delivered to the DAC.

e) Why is processing audio sample-by-sample a bad idea (short answer)?

3) Dynamic Programming (6 pts)

Fill in the matrix to compute the length of the best match of a sequence of performed notes to a sequence of notes from the score (the core algorithm for Score Following in Computer Accompaniment). If $P$ is the performed notes, $S$ is the score, and $M$ is an array, the update rule is:

$$M[r, c] = \max(M[r-1, c] - 1, M[r, c-1], (M[r-1, c-1] + 1) \text{ if } S[r] = P[c] \text{ else } 0),$$

i.e. penalize 1 for each skipped score note, no penalty for extra performed note, and credit 1 for each match.

Alternatively, for (substantial) partial credit, you can fill in $M$ to compute the Edit Distance between the two strings using:

$$M[r, c] = \min(M[r-1, c] + 1, M[r, c-1] + 1, M[r-1, c-1] + (0 \text{ if } S[r] = P[c] \text{ else } 2))$$

i.e. count 1 for each skipped score note and each extra performed note; mismatched notes count as both skipping and inserting (+2).

There are two arrays, but you need only complete one. The first row and column of each are filled in so you do not have to worry about boundary conditions:

For Score Following Best Match

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score (S):</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>-4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For Edit Distance

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score (S):</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
4) **Theory & Music Representation** (6 pts – 1 each)

![Musical notation image]

a) What is the key signature?

b) What is the time signature?

c) What is the MIDI key number (in decimal) for the first pitch that is played?

d) What is the MIDI key number (in decimal) for the lowest pitch played?

e) How many beats is the longest note?

f) How many beats is the shortest note?

5) **Networks and Music Control** (6 pts – 2 each)

a) O2 has two forms of send. \( \text{o2\_send()} \) sends a message as quickly as possible but does not ensure delivery. \( \text{o2\_send\_cmd()} \) 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:

\[ \text{o2\_send()} \quad \text{_________} \]

\[ \text{o2\_send\_cmd()} \quad \text{_________} \]

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.
6) **Standard MIDI Files** (6 pts – 3 each)
   The two main pieces of information in a Standard MIDI File are timestamps and events.

   a) Briefly describe SMF timestamps in terms of size (in bits or bytes), range of values, and unit of measure (i.e. seconds?, beats?, something else?)

   b) Briefly describe SMF events. What events are encoded and what is the encoding?

      Most SMF events are __________________________________________________________

      __________________________________________________________,

      for example, __________________________________________________________

      is a Standard MIDI File event.

7) **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.

8) **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:

![Diagram of clock synchronization]

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
      \[
      L_i = \text{[enter equation here]}
      \]
   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)
9) **Music Understanding and Classifiers** (10 pts – 3, 2, 2, 3)

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

In a Naïve Bayesian Classifier, we want to find the most likely class given some features.

a) Assuming one feature, and using the formula above, write an expression for \( P(\text{class}|\text{feature}) \):

\[
P(\text{class}|\text{feature}) = \frac{P(\text{feature}|\text{class})P(\text{class})}{P(\text{feature})}
\]

b) The classifier finds the value of class that maximizes both sides of this equality. Now, assume all classes are equally likely. How can we simplify the right hand side of your equation from part (a) to get a slightly simpler expression to optimize (show the new expression):

\[
P(\text{feature}) = \frac{1}{n}
\]

\[n\text{ is the number of classes.}\]

c) The remaining expression in (b) should have one or more terms that are independent of the class; thus you can simplify the expression that is to be maximized even further (show the new expression):

\[
P(\text{class}) = \frac{1}{n}
\]

\[n\text{ is the number of classes.}\]

d) Suppose the classes are Rock and Jazz and the feature is “beat strength.” And suppose you measure a “beat strength” feature that is 1 standard deviation away from the mean feature value for Rock, and 1.5 standard deviations away from the mean feature value for Jazz (we’re assuming feature values have a Gaussian distribution). What class should the classifier pick?
10) **Concurrency** (10 pts – 3, 5, 2)

Consider the following code that models a bank account with `withdraw()` and `deposit()` operations:

```c
#define SUCCESS 0 // define return status codes SUCCESS and FAIL
#define FAIL -1
lock mylock = lock_create(); // create a new lock object
int balance = 10; // a global variable: initial bank account balance
int withdraw(int amt) {
    if (balance >= amt) {
        balance = balance - amt;
        return SUCCESS;
    } else {
        return FAIL;
    }
}
int deposit(int amt) {
    balance = balance + amt;
    return SUCCESS;
}
```

a) Write a sequence of steps in two threads, where the initial `balance` is 10, thread A calls `withdraw(10)`, thread B calls `deposit(1)` and the final balance after both threads return `SUCCESS` is 11.

   THREAD A

   THREAD B
b) Modify the code above by adding calls to `lock(mylock)` and `unlock(mylock)` to obtain correct behavior when `deposit()` and `withdraw()` are called from multiple concurrent threads.

c) Priority inversion can occur when (pick the most specific answer that is true):
   i) There are multiple threads
   ii) There are at least three threads
   iii) There are at least three threads running at different priorities
   iv) There are at least three threads running at different priorities, and two of them need to acquire the same lock
   v) There are at least three threads running at different priorities, and two of them need to acquire the same lock, and the third thread becomes ready to run
   vi) All of (v), and the third thread needs to acquire the lock too
   vii) All of (vi), and the third thread is blocked waiting for the high priority thread
   viii) All of (vii), and deadlock occurs.

11) Matching (8 pts – 1 each)
Match each item at left with at most two corresponding items at right.

A. Visual Programming Language  ___ URL-like address
B. Algorithmic Composition techniques  ___ trie
C. Queries are 20s audio recordings  ___ music fingerprinting
D. O2  ___ sequence matching
E. Data structure for sequence learning  ___ Naïve Bayes
F. Dynamic Programming  ___ formal grammar
G. Genre Classification  ___ pattern generators
   ___ Pure Data (Pd)
12) **Event-based, real-time programming** (9 pts)
Assume you have a sensor that senses two states: 1 and 0. The function `sensor_get()` can be used to read the sensor, returning 1 or 0.

Write program in Serpent that polls the sensor every 20ms, starts a MIDI note with random pitch when the sensor changes to the 1 state, and turns the note off when the sensor changes to the 0 state.

Some useful function/method signatures (and you must use `rtsched.cause`) are:
- `rtsched.cause(time_delay, object, method, parm1, parm2, ...)`
- `midi_out.note_on(channel, keynum, velocity)`
- `pr_range(low, high) // choose random integer in [low, high−1]`

Please sketch the implementation on another page, then neatly copy the final version to be graded into place below:

```serpent
require "debug"  // always a good idea
require "sched"
require "prob"   // for pr_range()
require "midi-io"

// initialize any global variables here

// define start_sensing here to implement the behavior described above
def start_sensing():

sched_init()  // handy function to set up and start schedulers
```
13) **Logical Time** (5 pts)

Here is a naïve program to play 3 notes per second. It would be correct if computers were infinitely fast and accurate.

```python
while true
    play_note(60)  # assume this works
    time_sleep(1 / 3)
```

Rewrite this naïve program to play 3 notes per second with reasonable accuracy and no long-term drift. You can call `time_get()` to get the current real time as a double precision float in seconds. *Do not use a scheduler!*

**Suggestion:** Sketch the code on another page, then neatly copy the final version for grading here:
14) **Weinberg paper on Interconnected Music Networks** (4 pts)

Weinberg describes four approaches to interconnected music networks:

- a) The Server
- b) The Bridge
- c) The Shaper
- d) The Construction Kit

Label the following descriptions, excerpted from the paper, with the corresponding approach: (a), (b), (c), or (d).

- _____ System takes an active musical role by algorithmically generating musical materials and allowing participants to collaboratively modify these materials. Players can continuously listen and respond to the music that is modified by all participants. The approach does not support direct algorithmic interdependencies between players.

- _____ Interaction occurs independently in each player’s browser so that “each user can create his or her own unique experience.” Participants can only listen to their own creation, which significantly limits the sense of collaboration.

- _____ Players play and improvise as if they were in the same space. Participants can listen and respond to each other while playing. However, the role of the network in this approach is not to enhance and enrich collaboration, but to provide a technical solution for imitating traditional group collaboration.

- _____ Participants contribute music to multiple-user composition sessions, manipulate and shape their and other players’ music, and take part in a collective creation. Interaction is usually asynchronous as participants submit their pre-composed tracks and manipulate their peers’ material off-line.

15) **Music Generation and Algorithmic Composition** (4 pts)

- a) Name a limitation of Markov Chains for music generation.

- b) Other then Markov Chains, name two other approaches to Algorithmic Composition that we discussed in class.