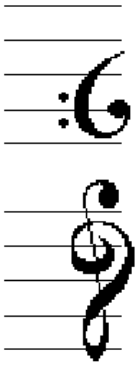
Name: _____ Andrew ID: _____

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

Spring 2019, *PRACTICE VERSION*

Useful Constants

MIDI number	Note name	Keyboard	Frequency Hz		Period ms	
21	A0		27.500		36.36	
22	B0		30.868	29.135	32.40	34.32
23	C1		32.703		30.58	
24	D1		36.708	34.648	27.24	28.86
25	E1		41.203	38.891	24.27	25.71
26	F1		43.654		22.91	
27	G1		48.999	46.249	20.41	21.62
28	A1		55.000	51.913	18.18	19.26
29	B1		61.735	58.270	16.20	17.16
30	C2		65.406		15.29	
31	D2		73.416	69.296	13.62	14.29
32	E2		82.407	77.782	12.13	12.86
33	F2		87.307		11.45	
34	G2		97.999	92.499	10.20	10.81
35	A2		110.00	103.83	9.091	9.631
36	B2		123.47	116.54	8.099	8.581
37	C3		130.81		7.645	
38	D3		146.83	138.59	6.811	7.216
39	E3		164.81	155.56	6.068	6.428
40	F3		174.61		5.727	
41	G3		196.00	185.00	5.102	5.405
42	A3		220.00	207.65	4.545	4.816
43	B3		246.94	233.08	4.050	4.290
44	**C4**		**261.63**		**3.822**	
45	D4		293.67	277.18	3.405	3.608
46	E4		329.63	311.13	3.034	3.214
47	F4		349.23		2.863	
48	G4		392.00	369.99	2.551	2.703
49	**A4**		**440.00**	415.30	**2.273**	2.408
50	B4		493.88	466.16	2.025	2.145
51	C5		523.25		1.910	
52	D5		587.33	554.37	1.703	1.804
53	E5		659.26	622.25	1.517	1.607
54	F5		698.46		1.432	
55	G5		783.99	739.99	1.276	1.351
56	A5		880.00	830.61	1.136	1.204
57	B5		987.77	932.33	1.012	1.073
58	C6		1046.5		0.9556	
59	D6		1174.7	1108.7	0.8513	0.9020
60	E6		1318.5	1244.5	0.7584	0.8034
61	F6		1396.9		0.7159	
62	G6		1568.0	1480.0	0.6378	0.6757
63	A6		1760.0	1661.2	0.5682	0.6020
64	B6		1975.5	1864.7	0.5062	0.5363
65	C7		2093.0		0.4778	
66	D7		2349.3	2217.5	0.4257	0.4510
67	E7		2637.0	2489.0	0.3792	0.4018
68	F7		2793.0		0.3580	
69	G7		3136.0	2960.0	0.3189	0.3378
70	A7		3520.0	3322.4	0.2841	0.3010
71	B7		3951.1	3729.3	0.2531	0.2681
72	C8	J. Wolfe, UNSW	4186.0		0.2389	



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

Channel Voice Messages [nnnn = 0-15 (MIDI Channel Number 1-16)]		
1000nnnn	0kkkkkkk 0vvvvvvv	Note Off event. This message is sent when a note is released (ended). (kkkkkkk) is the key (note) number. (vvvvvvv) is the velocity.
1001nnnn	0kkkkkkk 0vvvvvvv	Note On event. This message is sent when a note is depressed (start). (kkkkkkk) is the key (note) number. (vvvvvvv) is the velocity.
1010nnnn	0kkkkkkk 0vvvvvvv	Polyphonic Key Pressure (Aftertouch). This message is most often sent by pressing down on the key after it "bottoms out". (kkkkkkk) is the key (note) number. (vvvvvvv) is the pressure value.
1011nnnn	0ccccccc 0vvvvvvv	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 "Channel Mode Messages" (below). (ccccccc) is the controller number (0-119). (vvvvvvv) is the controller value (0-127).
1100nnnn	0ppppppp	Program Change. This message sent when the patch number changes. (ppppppp) is the new program number.
1101nnnn	0vvvvvvv	Channel Pressure (After-touch). This message is most often sent by pressing down on the key after it "bottoms out". This message is different from polyphonic after-touch. Use this message to send the single greatest pressure value (of all the current depressed keys). (vvvvvvv) is the pressure value.
1110nnnn	0lllllll 0mmmmmmm	Pitch Wheel Change. 0mmmmmmm 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. (mmmmmm) are the most significant 7 bits.

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

Control Number (2nd Byte Value)			Control Function	3rd Byte Value	
Decimal	Binary	Hex		Value	Used As
0	00000000	00	Bank Select	0-127	MSB
1	00000001	01	Modulation Wheel or Lever	0-127	MSB
2	00000010	02	Breath Controller	0-127	MSB
3	00000011	03	Undefined	0-127	MSB
4	00000100	04	Foot Controller	0-127	MSB
5	00000101	05	Portamento Time	0-127	MSB
6	00000110	06	Data Entry MSB	0-127	MSB
7	00000111	07	Channel Volume (formerly Main Volume)	0-127	MSB
8	00001000	08	Balance	0-127	MSB
9	00001001	09	Undefined	0-127	MSB
10	00001010	0A	Pan	0-127	MSB
11	00001011	0B	Expression Controller	0-127	MSB
12	00001100	0C	Effect Control 1	0-127	MSB
13	00001101	0D	Effect Control 2	0-127	MSB

Name: _____ Andrew ID: _____

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?

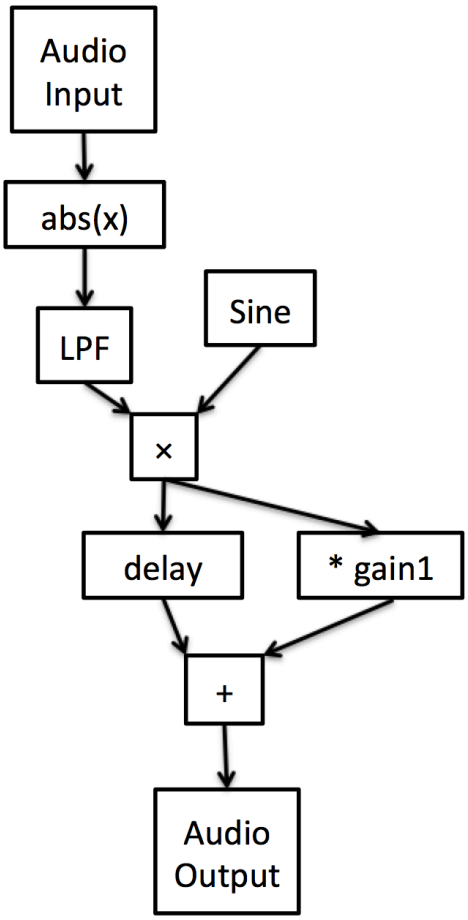
d) How many different instruments can you play simultaneously given a single conventional MIDI connection and synthesizer? (Ignore the special case of drum sounds.)

e) About how long (in milliseconds) does it take to send 10 note-on messages in MIDI? (Use B = 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

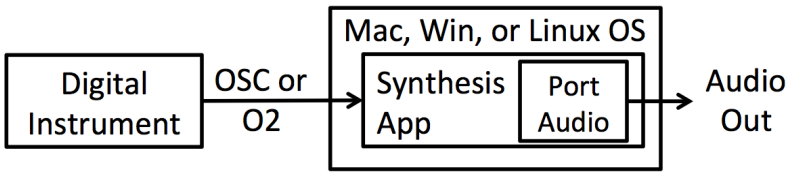
a) Label the following graph of unit generators with numbers to indicate the order in which they should be computed. An arrow from A to B indicates that data generated by A is read by B.



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.



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.

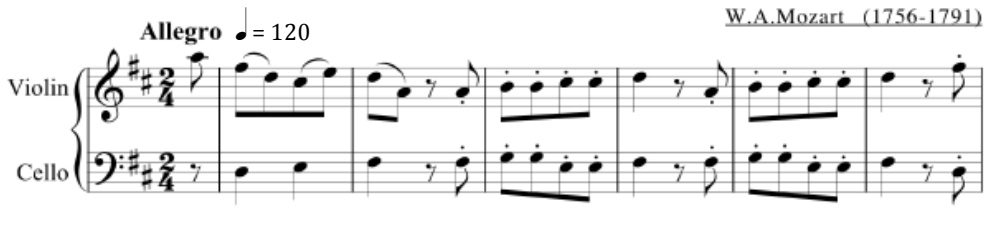
1) Music Generation and Algorithmic Composition (4 pts)

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

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

Name: _____ Andrew ID: _____

4) Theory & Music Representation



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

[*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.

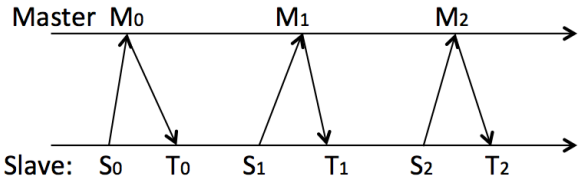
3) **Standard MIDI Files**

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:



1. Run steps (a) through (e) for $i = 0, 1, \ldots N\text{-}1$:

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 = _____.

e. The local clock offset is estimated as $\boldsymbol{\Delta_i} = L_i - M_i$.

2. Finally, the algorithm computes an overall estimated $\boldsymbol{\Delta}$ 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(Happy) = P(Sad) = 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

$$(|\text{observation} - \mu_{\text{classA}}|/\sigma_{\text{classA}}) < (|\text{observation} - \mu_{\text{classB}}|/\sigma_{\text{classB}})$$
$$\text{if and only if}$$
$$p(\text{observation} \mid \text{classA}) > p(\text{observation} \mid \text{classB}).$$

	Mean within class (μ_{class})	Standard Deviation within class (σ_{class})
Happy	65	4
Sad	57	8

a) If the observed *mean pitch* is 61 (half-way between μ_{happy} and μ_{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(Happy) = 1/4 and P(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
3 class Node:
4     var value
5     var next
6
7 def insert(value):
8     # Insert value at front of values.
9     var node = Node()
10    node.value = value
11    node.next = values
12    values = node
```

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.

Thread 1: `insert(1)`	Thread 2: `insert(2)`

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:

```
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):

```
note(pitch)
```

To schedule a function call, call (but do not define):

```
sched_cause(time_from_now, nil, function_name, p1, p2, …)
```

You can assume the scheduler in `sched.srp` is loaded and running in the normal way.

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
```

_____

_____

_____

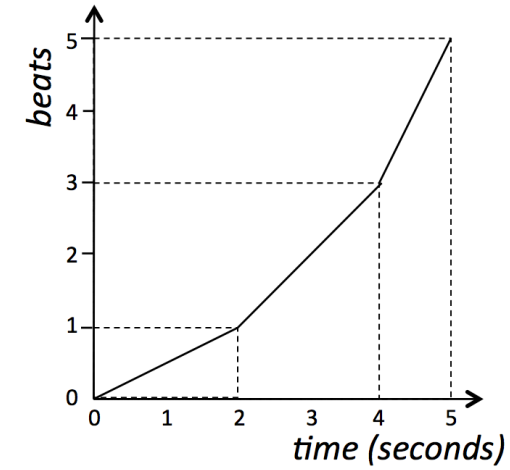
_____

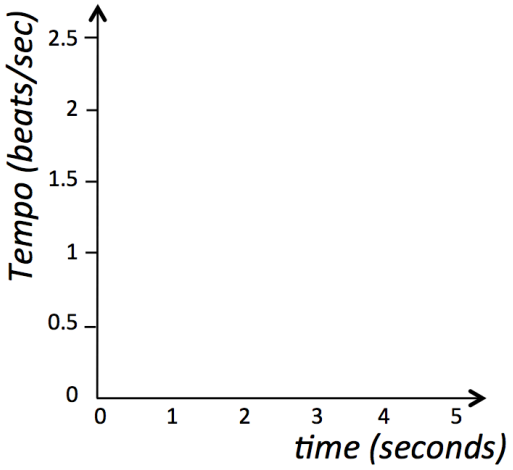
_____

// 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.