COS/MUS 314

Assignment 4: Arrays & Drum Machine

Assignment due 7 March 2012, 11:59 pm

This assignment is done with your partner.

Reading & Resources

Handout on arrays:
http://www.cs.princeton.edu/~fiebrink/314/2009/chuck_arrays.pdf
Handout on functions & concurrency (review):
http://www.cs.princeton.edu/~fiebrink/314/functions101.pdf
SndBuf references:
http://chuck.cs.princeton.edu/doc/program/ugen_full.html#sndbuf
and http://chuck.cs.princeton.edu/doc/examples/basic/sndbuf.ck
"On-the-fly" synchronization
Look at (and run!) the otf_01 to otf_08 examples at http://chuck.cs.princeton.edu/doc/examples/. Study how randomization is used, and the modulo (%) trick at the top of each file (see Question 2 below).

Question 1. Array practice (3 points)

a) Pick (or compose) a melody of at least 10 notes.

b) Declare two array variables:

- `notes` will contain the notes of your melody
- `durations` will contain the durations of each melody note
- You'll have to decide what type of arrays each should be (e.g., int, float, dur); there are multiple ways to do this.

c) Assign initial values to these two arrays, so that `notes` encodes the notes of your melody, and `durations` encodes the durations of each note. Note that the size of `notes` should be equal to the size of `durations`.

d) Create a synthesis patch (e.g., an oscillator, StkInstrument, or some of your custom synthesis code from assignment 3) to use to play your melody.

e) Insert into your code a for-loop of the form:

```
for (0 => int i; i < ???; i++) {
    //???
}
```

f) Edit the "???"s in the for-loop (i.e., the looping condition in the first line, and the loop body), so that each iteration through the loop will play one note of your melody. At this point, running your code should play the complete melody. Remember that if you are using an `Envelope`, you'll call .`keyOn` and .`keyOff` functions to ramp the envelope up and down. Also remember that you need to wait some time after `keyOff` is called before the sound decays to 0 gain, and you should take this into account without messing up the rhythm of your melody.

g) Now modify your code by putting the for-loop into a new function, e.g. called `playMelody`. Call `playMelody` in your code to play the melody (the result should sound the same as question 6).

h) Finally, modify your code so that it plays your melody in a round or canon (like "row row row your boat."). Hints:

- You'll need to make multiple copies of your oscillators and/or envelopes so that multiple voices can play at once.
- You'll need to parameterize your `playMelody` function to specify which oscillator/envelope to use.
- You'll need to spork one or more of your function calls. Advancing time in between function calls will delay each melody onset, creating a round.

Question 2: On-the-fly synchronization investigation (1 point)
Recall:

% is the modulo operator in ChucK. Read about it at http://chuck.cs.princeton.edu/doc/language/oper.html#mod if you're not familiar with this operator.

now is the number of samples that have passed since the virtual machine started running. (By default, miniAudicle will use a sample rate of 44,100 Hz.)

a) At what points in time will the expression

```
now % 44100::samp
```

be equal to

```
0::second ?
```

b) If T is a duration (e.g., initialized as `.5::second => dur T;`), what is the value of the expression

```
T - (now % T)
```

for an arbitrary value of now?

c) Explain in detail how this code snippet synchronizes multiple shreds that are added "manually" in miniAudicle:

```
.5::second => dur T;
T - (now % T) => now;
```

Question 3: Drum machine (6 points)
Create a drum machine with multiple parallel "voices" or "instruments" (at least three). When run, these parts will rhythmically complement each other in a musically interesting layered texture, like the "on-the-fly" drum examples linked in the Reading section. Not all parts actually have to be drums.

You have the option of putting each voice in a separate file which can be added by hand (like the on-the-fly examples, with modulo synchronization), putting each part in its own function in a single file (and calling/sporking those functions within that file), or (if you already know about user-defined classes) creating your own chuck class for each instrument, then providing at least one file that instantiates and uses objects of those classes.

Requirements:

- At least one voice/instrument should use pre-recorded samples. These could be the MiniAudicle default samples (e.g., kick.wav, within the data/ directory), sounds from freesound.org, and/or sounds you record yourself or find elsewhere.
- Include comments that tell us how to run/perform your drum machine
- Make something that you like!

Suggestions:

- If you use modulo synchronization, be sure that you choose the synchronization period **carefully** for every file!
- You can start with the drum machine skeleton at http://www.cs.princeton.edu/courses/archive/spring12/cos314/assignments/assignment4/skeleton.ck
- Use functions to make your life easier, much as in Assignment 3.
- Use arrays to make your life easier, much as in Question 1.
- Try incorporating randomization in useful ways to create a less mechanical sound, or to allow the structure to vary over time.
- Don't forget you can vary not only gain of a SndBuf, but also its rate and playback start position, for neat effects.

What to turn in:

- One chuck file containing your melody-playing code from Question 1.h. (If you end up substantially modifying your code between 1.g and 1.h, it's fine to turn in one ChucK file for 1.g and another for 1.h, and we'll grade them separately.)
- Your written answers to Questions 2.a, 2.b, and 2.c, included in a .txt, .doc, or .pdf file.
- Your drum machine for Question 3. Be sure to submit any sound files you use in your assignment, and please reference them in the code using a relative path (e.g., "data/kick.wav") so that the graders must simply change the path used by miniAudicle without having to edit your code or your directory structure. Please also include instructions for running your code in question 3, if it's not in just one file.