COS/MUS 314

Classes and Filters

Assignment due 6 April 2009

0. Reading:

@=>, =>, pointers, and memory

http://chuck.cs.princeton.edu/doc/language/oper.html#chuck
http://en.wikipedia.org/wiki/Pointer_(computing)
Class examples: http://www.cs.princeton.edu/~fiebrink/314/2009/week7/

Inheritance and polymorphism

Class examples: http://www.cs.princeton.edu/~fiebrink/314/2009/week7/
http://chuck.cs.princeton.edu/doc/language/class.html (especially section on Inheritance)

Filters

Example: http://chuck.cs.princeton.edu/doc/examples/basic/wind.ck
See filter section on
http://chuck.cs.princeton.edu/doc/program/ugen.html

1. Written question: @=> and references

Give at least two examples where @=> is necessary for assignment, as opposed to => . In general, for what types of variables is @=> necessary? What is the difference in the behavior of these operators?

2. Written question: Design & inheritance

Imagine you are a software designer, and you've been hired to work with a computer music performance ensemble. At your first rehearsal, the performers show up with a really cool new interface device, and they want you to write some code that let's them use this device to control a composition. You think, "Great! Sounds like fun!" You go home and write two classes: One, called CoolNewDevice, gets all the control messages from the device and saves a virtual representation of the device state (e.g., the current position of each slider, knob, joystick axis etc.). It also has some Event member variables that it broadcasts when buttons are pressed. The second class, called Composition, has a member variable that is an instance of a CoolNewDevice object; it uses the state of this object (e.g., each slider position) to set its parameters (e.g., volume) in real-time, and it listens for the object's events in order to respond to button presses and other changes.

You show up at the next rehearsal very proud of your ingenious use of object-oriented design. Unfortunately, you soon come to realize that your performers, though wonderful people, are a bit unstable. In fact, every week, they show up to rehearsal with a new, even weirder, controller device, and they expect you to change the code so that this device will work with their composition. How will you modify the design of your code in order to minimize the amount that you have to re-write every time they bring you a new controller? Describe what classes you will write, how they will work together, and how you can use inheritance to make your life easier. (Don’t write any code.)

3. Filters
*****HEALTH & SAFTEY WARNING! Be careful with filter parameters, especially reson Q values (set Q > 1; higher Q means a sharper filter) and pole radii (always set radius < 1). Do this assignment with your volume down & headphones off! And make sure to write the interactive portion of the code so that you’ll never get undesirable values accidentally!*****

Pick two ChucK filters (that extend FilterBasic). Write a ChucK file (or two files, one for each filter) that filters some sound input (from the adc, a sample, a UGen, your choice). Hook up one or more parameters each filter so that you can control them expressively in real time using the mouse, keyboard, motion sensor, or other input device.

Describe what is happening when you change the filter parameters for each filter, and how and why this affects the sound that you hear.

4. PRC **2-point bonus** question on filtering:
Pick one (or both) of these two Fun Filter Frolics:

A) Math/ChucK NERD:
Implement a resonant (two pole) filter directly in ChucK (use a 1.0::samp=>now update). The "difference equation" for a two-pole filter is:

y(n) = b0*x(n)-a1*y(n-1)-a2*y(n-2)

Compare performance to the built-in ChucK TwoPole UG. Make sure they sound the same (We suggest using Noise as an input). Comment.

Note: Any DSP operation (including those of existing unit generators) can be implemented in native ChucK. The power is in being able to implement something that ChucK doesn't provide a unit generator for. For example, to hear an impulse each time an input mic/line signal is greater than 0.99, we could:

```
adc => blackhole;
Impulse i => dac;

while (1)   {
   1 :: samp => now;
   if (adc.last() > 0.99) 1.0 => i.next;
}
```

This example is of questionable value, but it lets you know how one might implement any function at the sample rate in ChucK.

B) Composer NERD:
Listen to Richard Karpen's "Exchange"
http://www.cs.princeton.edu/courses/archive/spr09/cos325/music/ExchangeRichardKarpen1987.mp3
This piece uses resonant filters to create a tape accompaniment for a flute, controlling the filter resonance right up to and beyond stability (radius = 1+epsilon). Do something interesting along these lines, but explain what you're doing.

What to hand in:

- Your written responses to questions 1 and 2
- Your code and written response for question 3.
- Optionally your response/code for question 4.