Review of ChucK basics

COS/MUS 314, 2/2/10

The => operator

The "=>" operator ("chuck operator") is used for:

1. Connecting UGens in a synthesis patch:

```
SinOsc s => dac;
SawOsc s2 => Gain g => Envelope e;
Etc.
```

Here, the audio samples output by the UGen on the left of "=>" is input into the UGen on the right.

2. Assigning values to variables:

```
3.5 => float f; //f is now 3.5
float g;
f => g; //now g is also 3.5
```

Think of this as taking the value on the left (or the value of the variable on the left) and "chucking" it at the variable on the right, assigning the variable on the right a new value.

3. Passing arguments to functions

```
500 => s.freq; //same as calling s.freq(500); sets frequency to 500Hz
```

4. "Advancing time"

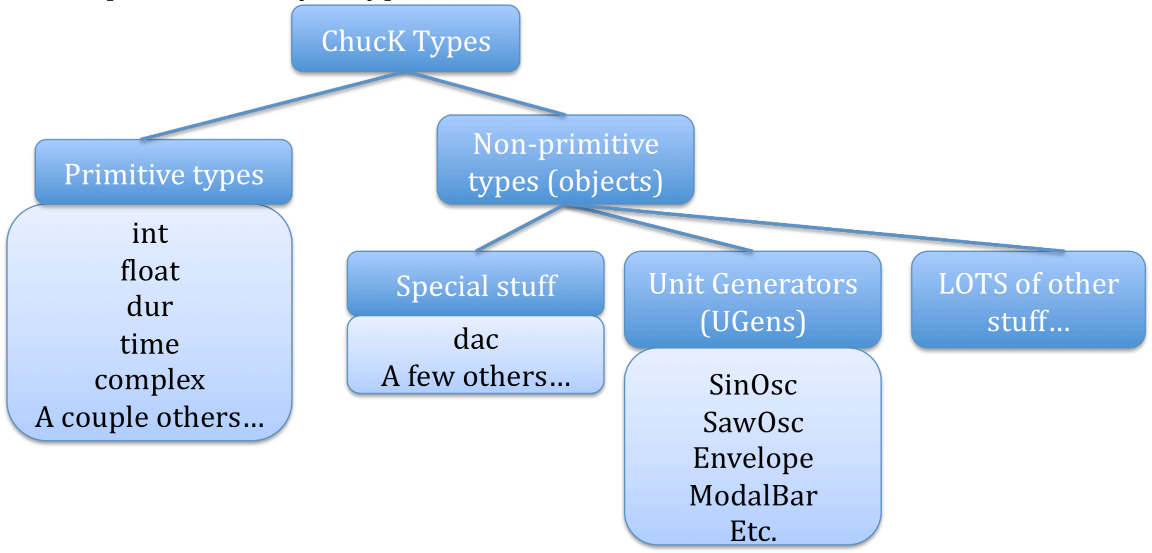
Any line of code that ends with "=> now; " tells ChucK to stop executing lines of code and just make sound for a certain amount of time:

```
330 => s.freq;
531::second => now;
500 => s.freq;
```

This tells ChucK to wait 330 seconds between setting s's frequency to 330 and setting it to 500. See the section below on "Time in Chuck"

Types

A conceptual hierarchy of types in ChucK:



Primitive types are the simplest types of values you can represent in the language.

- Numbers are represented as int (for positive or negative integers; no decimal places allowed), float (for any real numbers), or complex (for complex numbers: we won't be using these just yet).
- Durations in time are represented as dur values. To specify a dur value, you need to specify a legal unit, using the following syntax:
    - 1.0::second, 300::second, etc.
    - 35::ms
    - 31::samp (samples: i.e. 1/44100th of a second using a standard sample rate)
    - 4::hour
    - etc.
- Moments in time are represented by the type time
    - "now" is a special variable denoting this moment in time, right now
    - now + 5::second => time later;
        - This creates a time variable called later, which represents the time 5 seconds in the future from now

Objects typically have **functions** defined that you can call. Primitive types never have functions of their own. Objects also often have multiple "properties" associated with them; for example, oscillators like SinOsc all have a frequency value, which is itself represented as a float.

Declaration and instantiation

Regardless of the type (primitive or not), you declare a variable in chuck using the following syntax:

typeName variableName;

e.g.

```
SinOsc s;
int i;
```

Once you've declared a variable, you can use it; it's instantiated to some default value (for a primitive) or state (for an object). For example, above, i has the value 0, and s has a default frequency of 440.

You can also instantiate a primitive to a different value at the time of declaration, using the => assignment operator:

```
1 => int b;
2.5 => float f;
```

Functions

Functions are used to:

1. Get the current value of some property of an object

   ```
   SinOsc s;
   s.freq() => float f;
   ```

   Here we call the .freq function with empty parentheses to indicate that we want s's current frequency; we store the result in f. Note that => functions as an assignment operator here (usage #2 of "=>" above).

2. Change a property of an object

   ```
   440 => s.freq;
   ```
   or
   ```
   1000 => float f;
   f => s.freq;
   ```
   Here we are setting the value of s's frequency to a value. We're using "=>" as usage #3 of "=>" above, to send a parameter (the desired new frequency) to the function.

3. To do something else (e.g., compute a value, or trigger an action) (see below)

UGen functions

To discover what functions are available to call on a UGen object to get information or change a property, see http://chuck.cs.princeton.edu/doc/program/ugen.html This documentation will tell you about the functions available, as well as their *types*.

For example, under SinOsc, we see:

"• **.freq** - (float , READ/WRITE) - oscillator frequency (Hz), phase-matched"

This tells us that every SinOsc object has a .freq function. We can get the frequency using .freq() (because it says "READ") and we can set the frequency using something like

```
430 => s.freq;
```

because it says "WRITE").

Gain

Every UGen also has a .gain function (float, READ/WRITE), which allows you to scale the gain (amplitude) of its output. For example,

```
.01 => s.gain;
s.gain() => float myGain;
```

Special functions for computing stuff

A few special functions exist for our convenience. They're not associated with objects, per se; we can call them at any time. For example:

- Std.mtof(60) => float f;
    - mtof converts MIDI note values (60 = middle C) into frequency values in Hz (which are expected by SinOsc and other UGens)
- Std.ftom(440) => float m;
    - ftom does the reverse
- Std.rand() => int i;   //generate a random int
- Std.randf() => float f;//generate a random float
- Std.rand2(100, 1000) => int i; //generate a random int between 2 numbers
- Std.rand2f(0, 1) => float f; //generate a random float between 2 numbers

A list of these functions is available at
http://chuck.cs.princeton.edu/doc/program/stdlib.html

Note that "Std" denotes the "standard" set (namespace) of globally available functions, and you always have to use it when calling the function.

Time in ChucK

In general, each line of code in your ChucK program will execute in order from top to bottom, as fast as the computer can execute (hopefully pretty fast). The exception is lines of code that "chuck" something to "now", for example:

```
SinOsc s => dac;
440 => s.freq;
1::second => now;
880 => s.freq;
```

Above, 1 second will pass between setting s's frequency to 440 and setting it to 880. During that second, the dac will be grabbing output from s and playing it out the speakers, and because s's frequency has been set to 440, you'll hear a 440Hz sine wave.

In contrast, the following code doesn't "advance time" by chucking something to "now" in between setting the frequency to 440 and setting it to 880:

```
SinOsc s => dac;
440 => s.freq;
880 => s.freq;
```

so you'll never hear s playing at 440Hz.

Misc. stuff

A **comment** is a bit of text that doesn't get executed: it's a note to yourself and future users of your code. Anything after "//" on a single line is a comment:

```
SinOsc s => dac; //The stuff to the left was code but this is a comment!
```

You can make multi-line comments by surrounding your comment with /* … */

```
/* For example
 I could stick all this stuff
here
in the code
and it would be a long comment. */
```

To **print** out something to the miniAudicle console, surround the thing you want to print with <<< and >>>:

```
<<< "Hello!">>>;
<<< s.freq() >>>;  //print out frequency of s
<<< "My frequency is ", s.freq() >>>;
```

Use a comma to concatenate values you print out, like in the last line above.