ChucK Assignment 2

5 April 2012

In this lesson you will learn about variables, conditional tests, and how to perform repetitive tasks automatically. You'll also explore the properties of the natural harmonic series.

1 Variables

1. We're going to continue to use oscillators for this lesson. Review the techniques from lesson 1 for connecting and controlling oscillators.

2. Sometimes we need to be able to store and recall numbers, times, and other pieces of information. This is what **variables** are for. Do you remember variables from high school algebra? Well, forget them! The concept of variables in computers is completely different.

   - Variables are just *containers*. Each container holds one thing. That thing might be a number, or a duration, or a piece of text, or even an oscillator.
   - In order to use a variable, you first must create it, give it a name, and indicate what kind of information it will hold.
   - The three most common variable types are **int**, **float**, and **dur**, representing integers, floating-point numbers, and durations. We'll discuss more about the differences between these later.

   Here is the code to create a new integer variable called **myNum**:

   ```
   int myNum;
   ```

   Now we can store a value in that variable:

   ```
   25 => myNum;
   ```

   That variable can now be used in place of a simple number in your code:

   ```
   TriOsc myOsc => dac;
   myNum => myOsc.freq;
   ```

3. Here's the cool twist: variables can be changed, and you can use the *old* value of a variable to set the *new* value.

```
int myFrequency;
55 => myFrequency;
<<< myFrequency >>>; // This prints the value on screen
myFrequency + 100 => myFrequency;
<<< myFrequency >>>; // Now it's 155
```

2 Conditional Tests

1. Now that we have numbers in variables, we sometimes want to use them to make decisions. For this we use the **if** command:

```
float myGain;
0.8 => myGain;
if (myGain > 1)
{
  <<< "myGain is greater than 1">>>;
} else
{
  <<< "myGain is not greater than 1">>>;
}
```

2. The **conditional test** is the part inside the parentheses. In this case, myGain is 0.8, so the test *is myGain greater than 1?* is false.

3. The following are possible conditional tests:

```
x > y          is x greater than y?
x < y          is x less than y?
x >= y         is x greater than or equal to y?
x <= y         is x less than or equal to y?
x == y         does x equal y? (notice 2 equal signs)
x != y         is x not equal to y?
```

3 Repetition

1. If you want to do something many times, you could just copy and paste it in your code, but that's not a very efficient way of working. There are several methods of doing repeated things in ChucK. The first is the **repeat** command.

```
repeat(10)
{
  <<<"Hi, nice to meet you! I have no short term memory!">>>;
}
```

2. We can use repeat to create a series or pattern by changing a variable in every repetition.

```
int myNumber;
5 => myNumber;
repeat(10)
{
  <<< myNumber >>>;
  myNumber + 1 => myNumber; // increase it by 1
}
```

There is a useful shortcut for increasing a number by 1: myNumber++

3. You are probably familiar with the natural harmonic series. It is produced by playing frequencies that are all multiples of the same fundamental.

```
TriOsc myOsc => dac;
0.5 => myOsc.gain;
int fundamental;
int harmonic;
55 => fundamental;
1 => harmonic;
repeat(10)
{
  fundamental * harmonic => myOsc.freq; // multiply
  <<< "frequency:",fundamental * harmonic >>>; // print it
  harmonic++; // increase by 1
  250::ms => now;
}
```

EXTRA CREDIT: There are multiple methods of creating the harmonic series. Rewrite the above code to do the same thing but using a different method of calculating the frequency.

4. More complicated than **repeat**, but much more flexible, is the **for** loop. A **for** loop contains a conditional test, so you can decide whether you want to keep looping or not.

   A **for** loop looks like this:

```
for(int num; num < 10; num++)
{
  <<< num >>>;
}
```

See the three different parts in the parentheses?

```
        1          2         3
for(int num; num < 10; num++)
```

- The first part is the *initialization*. This is something done just before it begins looping. In this case, a new **int** called num is created.
- The second part is the *conditional test*. The commands in the brackets will be followed it it's true. If it's false, the looping ends.
- The third part is the *enumeration*. Simply put, it's what happens at the end of every repetition.

You can remember the form of a for loop with the mnemonic ICE:

```
I: Initialization
C: Conditional test
E: End of Every repetition
```

5. Here is the same harmonic series code as above, but now using a **for** loop instead of **repeat**:

```
TriOsc myOsc => dac;
0.5 => myOsc.gain;
int fundamental;
55 => fundamental;
for (1 => int harmonic; harmonic < 11; harmonic++ )
{
  fundamental * harmonic => myOsc.freq;
  250::ms => now;
}
```

Assignment 2: Harmonic Series Etude

Experiment with the harmonic series code above. Using variables and a repeat() or for() loop, create a short etude that explores the harmonic series. Be creative: try to go beyond the simple idea of an ascending arpeggio. Save your ChucK code as a .ck or .txt file and upload it to the assignment page on Blackboard by this Tuesday, April 10.