16 February 2008
COS/MUS 314

While loops and for loops in ChucK

For more information, see http://chuck.cs.princeton.edu/doc/language/ctrl.html

While loops

All while loops have the form:

```
while (condition) {
        do stuff
}
```

where *condition* is a Boolean expression that evaluates to either true or false, and *do stuff* is some code that does something.

For example:

```
0 => int i;
while (i < 10) {
        <<< i >>>;
        i++; //same as i + 1 => i;
}
```

Here, the condition is "i < 10", which will evaluate to true if i is less than 10 and false if i is 10 or higher. The body of the loop prints out the current value of i and uses the ++ operator shorthand to increment i by 1.

Boolean variables and expressions in chuck

For more information, see
http://chuck.cs.princeton.edu/doc/language/oper.html#log

Boolean logic deals with evaluating expressions that are either true or false. Generally speaking, a Boolean variable stores the value true or false, and a Boolean expression is a combination of Boolean variables and logic operations (and, or, not, is-greater-than, is-equal-to, etc.) that evaluates to true or false for particular values of the variables in the expression.

In chuck, **Boolean variables** are really just integers, where 0 is equivalent to false and 1 is equivalent to true. (In fact, any nonzero value will evaluate to true).

I can write the following statements:

```
if (0) {
        <<< "0 is true">>>;
} else {
        <<< "0 is false">>>;
}

if (1) {
        <<< "1 is true">>>;
}

if (335) {
        <<< "335 is true" >>>;
}
```

So, if I want to declare a variable to use as a Boolean, I just declare it as an int.

```
1 => int b;
```

You can also use the keywords true and false, which are in fact just the integers 1 and 0.

For example,

```
if (true) {
        <<< "True is true">>>;
}

<<< "True plus true equals:">>>;
<<< true + true >>>;

<<< "True plus false equals: ">>>;
<<< true + false >>>;
```

Boolean expressions employ the ChucK logic operators, which are the same as in C++, Java, and many other languages. See a list at http://chuck.cs.princeton.edu/doc/language/oper.html#log.

Expressions with these operators can be combined, and you can use parentheses to specify how they will be evaluated.

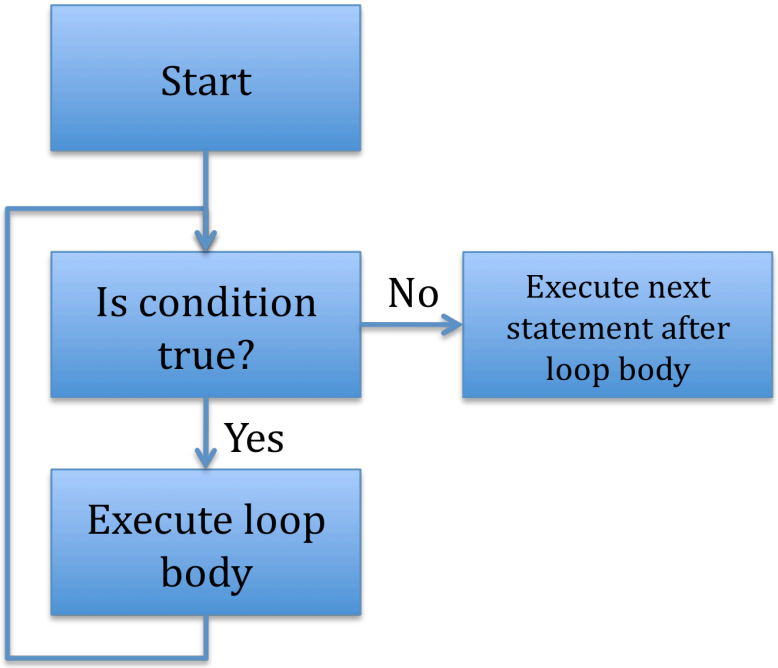
```
3 => int a;
5 => int b;
9 => int c;

//Compare the following two if-statements:
if ((c < 10 || a == 3) && b != 5) {
        <<< "First true">>>;
}
```

```
if (c < 10 || (a == 3 && b != 5)) {
        <<< "Second true">>>;
}
```

Hint: in both of the above,
(c < 10) is true, (a == 3) is true, and (b !=5) is false.

Execution of while loops



For-loops
A for loop has the structure

```
for (initialization ; condition ; expression) {
        do stuff
}
```

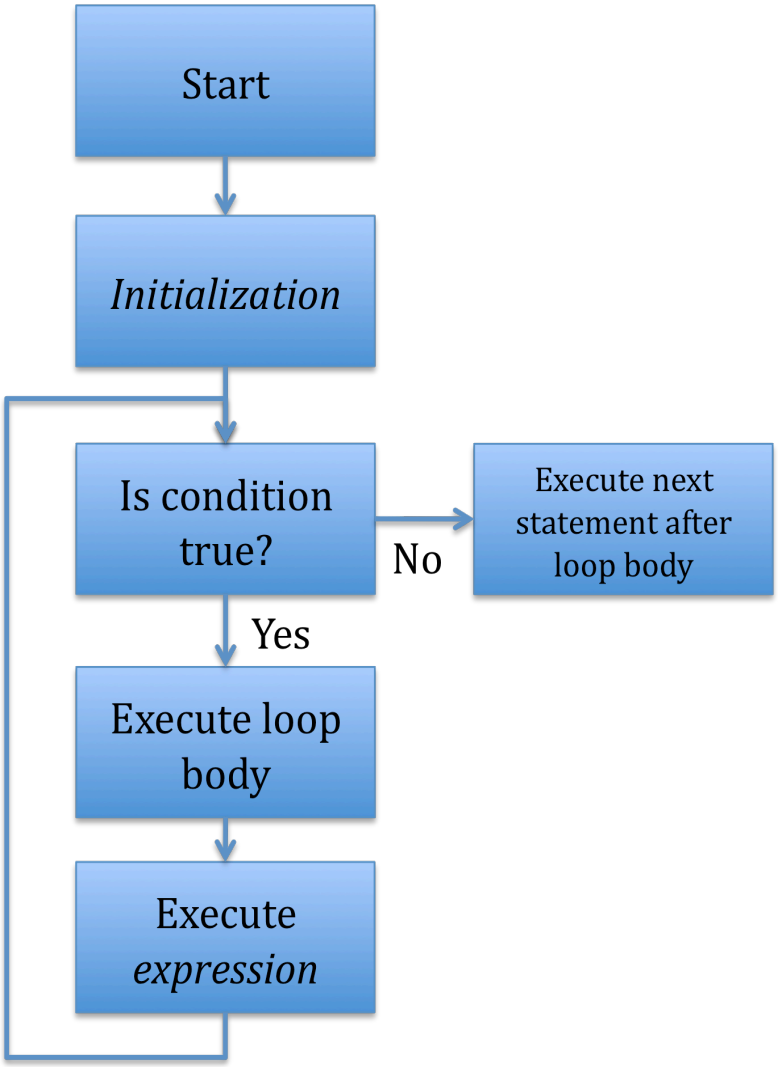
The condition and loop body play the same rolls as in while loops. Here, though, the *initialization* is typically used to initialize a "counter" variable before the first iteration of the loop, and the *expression* is typically used to increment this counter after each loop iteration.

For-loops can accomplish exactly the same set of tasks as while-loops, but they're most useful (and most often used) to execute something a set number of times. (Leaving out the initialization and expression is equivalent to using a while loop.)

For example, this prints out the numbers from 1 to 10

```
for (0 => int i; i < 10; i++) {
        <<< i >>>;
}
```

The for-loop execution path:



Loops and arrays

For-loops can be used easily with arrays to perform an operation on each array element.

For example:

```
//Print out the elements of an array
[10,20,40,22] @=> int x[];
for (0 => int j; j < x.size(); j++) {
        <<< x[j]>>>;
}
```

A more interesting example:

```
//Play a rich harmonic spectrum using a number of sine oscillators at harmonic
//frequencies
SinOsc s[3]; //vary # oscillators to change timbre

for (0 => int j; j < s.size(); j++) {
        s[j] => dac;
        440 * j => s[j].freq;}

1::second => now;
```

Another example:

```
//Play twinkle
[60,60,67,67,69,69,67,65,65,64,64,62,62,60] @=> int notes[];
[1,1,1,1,1,1,2,1,1,1,1,1,2] @=> int beats[];

Mandolin m => dac;
.25::second => dur quarterNote;

for (0 => int i; i < notes.size(); i++) {
        Std.mtof(notes[i]) => m.freq;
        1 => m.noteOn;
        beats[i]::quarterNote => now;
        1 => m.noteOff;
}
```