16 February 2009
COS/MUS 314

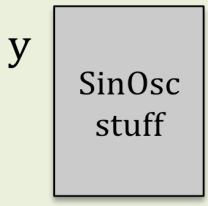
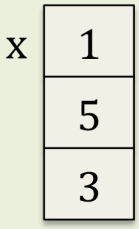
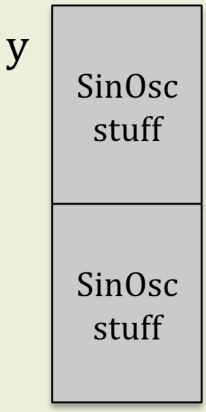
Arrays in ChucK

See also http://chuck.cs.princeton.edu/doc/language/array.html

What is an array?

An array is a list of objects of the same type (e.g., a list of ints, a list of floats, or a list of SinOscs). There may be 0 or more objects in the array.

Single variable	Array
Declaring without assigning a value The following code creates a variable x with the default value 0 (an int) and a variable y with the default value 0.0 (a float). int x; float y;	The following code creates a variable x that is an array of 1 integer (a list containing only one element, which is 0), and a variable y that is an array of 10 floats (a list containing 10 elements, each of which is 0.0). You **must** specify the size of the array when you declare it like this, though the size can be 0 (an empty array with no elements in it). The elements of the array will be instantiated to the default values for their types (e.g., 0 for int, 0.0 for float). int x[1]; float y[10];
Declaring and assigning a value (instantiation) at the same time The following code creates a variable x with the value 0 and a variable y with the value 10.5. 0 => int x; 10.0 => float y;	The following code creates an array variable x with 5 integer elements (0, 1, 3, 510, 23), and an array variable y with 2 float elements (10.92, 0.0). When instantiating and declaring arrays in this manner, you **must not** indicate the size of the array. The size will automatically be set based on the size of the list on the left. You **must** also use @=> for assignment instead of => , since an array is not a primitive type. [0,1,3,510,23] @=> int x[]; [10.92, 0.0] @=> float y[];
Assigning a new value The following code initially assigns 0 to the value of the int variable x. The second line changes x's value to 10. 0 => int x;	The following code initially assigns x to be an integer array with the contents (1, 2). The second line assigns x to be a new integer array with the contents (3, 4, 5). Note that we can reassign x to be an integer array of a different length, but because x is declared to be of type integer array (the bold part on the first line), we **can't** assign x to be a float

10 => x;	array or an array of other types (as in the commented out part labeled "illegal"). [1,2] @=> **int x[];** [3,4,5] @=> x; //illegal: //[3.2, 1.0] @=> x;
Memory representation Any variable—whether it is a primitive type, an object, or an array—can be understood as placing a label (e.g., "x") on a particular chunk of memory. The size of this chunk of memory is determined by the type of value stored there (e.g., it takes more memory to store a float than an int). A non-array variable can be understood as placing a label on a chunk of memory of the appropriate size for its type. 5 => int x; results in:  SinOsc y; results in: 	An array variable can be understood as placing a label on a contiguous series of memory chunks of the appropriate size. There will be one chunk for each array element. [1,5,3] @=> int x[]; results in:  SinOsc s[2]; results in: 
Using variables Primitive types require nothing special. 10 => int x; 15 + x => x; <<< x>>>;	You can assign an array variable using @=> as above. But if you want to access the individual elements (e.g., the int values themselves), you have to use an index. This index indicates the "chunk number" of the element we want in the array, starting with 0. That is, x[0] is always the first element in the array, and x[*n*-1] is the last for an array of size *n*.

SinOsc s;
10 => s.freq;
	[1,3,5] @=> int x[];
	<<< x[0] >>>; //Prints out 10
	15 + x[1] => x[2]; //Assigns 3rd value of x to be 3 +
	//15 = 18.
	
	SinOsc s[2];
	10 => s[0].freq;
	
	Note that we use => instead of @=> for assignment when we are dealing with primitive types, like the elements of x or the frequency value of s[0].

Misc. notes	You can get the size of an array by using the .size() method. For now, we're going to be dealing with static arrays, so use .size() just to get the size and not to set it.
	
	[1,4,10] @=> int x[];
	<<< x.size() >>>; //prints out 3
	100 => x[x.size()-1]; //sets the last element of x to be 100 instead of 10
	
	If you try to access an array element that doesn't exist (with an index that is too high), ChucK will crash and print out a message "ArrayOutOfBounds."