Arreglos: ordenar y acceder a los datos de tu composición

Este capitulo cubre:

- Declaración e inicialización de arreglos
- Recuperación y modificación de datos en arreglos
- Almacenamiento de diferentes tipos de datos en arreglos
- Uso de arreglos para controlar parámetros musicales en una canción

Ahora que has sido introducido al poder de las utiliidades de las bibliotecas Standard y Math, vas a aprender una de las cosas más importantes para hacer que el código de tus melodías y composiciones sea mucho más fácil y expresivo y te permitirá crear canciones de mucho mayor duración evitando escritura excesiva. ESte capítulo cubre arreglos, que son mecanismos superpoderosos que te permiten hacer colecciones de datos que puedes usar para variadas cosas. Los arreglos son fundamentales al funcionamiento de casi todos los programas de computadores. Son la forman en que los correos electrónicos, imágenes, archivos de sonido/música y todo lo demás son almacenados y accedidos dentro de computadores, teléfonos inteligentes, tablets y en la web.

Un arreglo es una colección de datos grabados en la memoria. Para los compositores-programadores, un uso de los arreglos es almacenar listas de parámetros musicales que quieres usar para controlar tus canciones a lo largo del tiempo, como alturas musicales (o números de notas MIDI), volúmenes, tiempos y/o duraciones. Otro uso para los arreglos es almacenar listas de strings como letras de canciones. Los arreglos pueden

contener cualquier tipo de dato, incluso puedes hacer areglos de unidades generadoras como SinOsc o Noise, cosa que harás en el siguiente capítulo.

Hasta el momento has generado melodías usando bucles para aumetnar o disminuir frecuencias o alturas, o has hechos programas largos que especifican cada nota que deseas, línea a línea. ¿No sería genial si pudieras hacer al comienzo del programa una lista de todas las notas que quieres tocar? ¿Y si pudieras de alguna forma almacenar tus parámetros de composición en la memoria, haciendo más fácil la escritura de canciones completas? Eso es exactamente lo que los arreglos te permiten hacer.

3.1 Declaración y almacenamiento de datos en arreglos

Aquí empezarás a aprender sobre cómo tocar notas de nuestra canción "Twinkle" mediante el almacenamiento de ellas en un solo arreglo. En este caso, grabarás tu melodía como una lista de números de notas MIDI. Para empezar, visualicemos conceptualmente qué es un arreglo.

En la figura 3.1 puedes ver siete celdas que contienen enteros. Estas celdas son siete trozos de memoria adjuntados a tu arreglo. Dentro de las celdas tenemos números enteros; en este caso ellos representan las notas MIDI de nuestra melodía "Twinkle". Bajo cada celda, puedes ver números que corresponden al índice de cada celda. Observa que los índices empiezan en 0. Todos los arreglos empiezan con el índice 0. El índice 3 del arreglo te dará el dato en la celda, en este caso, el número 64 de nota MIDI, la cuarta nota de Twinkle.

Ahora vemaos cómo representar este concepto con código. Representas un arreglo con paréntesis cuadrados [], con el número dentro de los paréntesis cuadrados siendo el número de elementos en el arreglo. En este caso hay sietes notas MIDI, así que declaras un arreglo a como int a[7] (1) en el siguiente listado. Observa que el arreglo es de tipo entero (puedes hacer arreglos de cualquier tipo, como verás más adelante). A continuación, tienes que almacenar tu melodía en el arreglo. Esto lo haces poniendo cada entero en su ubicación correspondiente en el arreglo, uno a la vez, como se ve en las líneas entre (2) y (3).

Listado 3.1 Declarando y llenando de forma larga un arreglo de enteros

```
//delaración de arreglo (método 1)
//(1) declara un arreglo de un largo específico (7)
int a[7];

//(2) define el valor almacenado en el elemento cero
57 => a[0];
57 => a[1];
64 => a[2];
64 => a[3];
66 => a[4];
66 => a[5];
64 => a[6];
//(3) define el valor almacenado en el último elemento

<<< a[0], a[1], a[2], a[3], a[4], a[5], a[6] >>>
```

Ahora tienes tus notas MIDI en el arreglo, pero esto tomó muchas líneas de código. ¿Existe un atajo? ¡Por supuesto que sí! Como se muestra en el listado 3.2, puedes grabar todas tus notas MIDI en una sola línea de código. Observa que ahora usamos un nuevo operador @=>, que es un

operador especial de ChucK usado para almacenar todos los datos en tu arreglo a, de una vez. @=> es llamado el operador at-ChucK o el operador de asignación explícita. Puedes usar este operador para copiar otros objetos en ChucK, pero por lejos su uso más común es para copiar listas de elementos a arreglos.

Listado 3.2 Declarar e inicializar un arreglo de una vez

```
[57, 57, 64, 64, 66, 66, 64] @=> int a[];

<<< a[0], a[1], a[2], a[3], a[4], a[5], a[6] >>>
```

La otra parte de esta nueva manera de declarar e inicializar un arrelgo es que ahora la declaración de tu arreglo a no necesita el [7] preasignado como número de elementos. Esta es la parte que el operador @=> te permite hacer. El arreglo a apunta al comienzo de tu lista de notas MIDI, pero el tamaño no necesita ser determinado durante la declaración. ChucK se encarga de esto por ti, haciendo fácil añadir y restar notas para hacer que la melodía sea más larga o corta, tecleando números extra en la lista, sin tener que contar cuántos notas hay cada vez.

3.2 Grabar y modificar datos en arreglos

AHora sabes un par de maneras de poner datos en arreglos: un elemento a la vez o copiando la lista a través de @=>, el operador at-ChucK. ¿Pero cómo accedes a estos datos? El listado 3.3 muestra cómo hacerlo. Aquí puedes leer un elemento del arreglo en la lína (1) y luego imprimir el resultado en (2). ¿Pero qué imprimie esto? Piénsalo. ¿Dijiste 57? ¡Recuerda que el índice empieza en 0! La respuesta correcta es 64 (el tercer item de la

lista).

EMPEZAR EN CERO Podrías estar preguntándote por qué este libro empezó con el capítulo 0 en vez del capítulo 1. Esto es porque somos programadores, y queríamos que empezaras a pensar dedse el principio en cosas como los arreglos de los índices empezando en cero en vez de uno. Dato interesante: muchos edificios de ciencias de la computación en universidades nombran el piso base como piso cero.

¿Qué pasa si quieres cambiar la melodía desde dentro de tu programa? No es un problema; un arreglo contiene variables, así que puedes cambiar lo que está en el arreglo en cualquier momento, definiendo un nuevo entero en la ubicación a[2] del arreglo (o cualquier otra ubicación) (3).

Listado 3.3 Acceso (lectura y escritura) a datos en un arreglo

```
//Declara e inicializa un arreglo
[57, 57, 64, 64, 66, 66, 64] @=> int a[];

//lectura del arreglo según índice
//(1) busca la nota en el arreglo según un índice entero
a[2] => int myNote;

//(2) lo imprime
<<< myNote >>>;

//¿quieres modificar los datos? ¡cero problema! (imprime tambi
//(3) cambia el valor elemento del arreglo según índice
61 => a[2];
<<< myNote, a[2] >>>;

<<< a[0], a[1], a[2], a[3], a[4], a[5], a[6] >>>
```

Este código debe imprimir en la ventana Console

```
64 :(int)
64 61
```

Esta salida refleja el hecho de que cambiaste los datos en un elemento de tu arreglo (a 61), pero el dato que previamente habías leído ahí y almacenado en la variable myNote no cambia (todavía es 64). La variable myNote está almacenada de forma separada que el dato del arreglo, así que puedes cambiar uno sin afectar al otro.

Puedes estar preguntándote ahora. "¿Que pasa si cambio el tamaño del arreglo después de declararlo?" Es posible hacerlo, como lo describimos en detalle en el apéndice B, sección B.6. Pero eso es un poco avanzado para nuestra tarea actual, así que prosigamos.

3.3 Usar datos de arreglos para tocar una melodía

Ya sabes cómo crear y llenar arrglos, así que ahora finalmente puedes usar tu arreglo para controlar sonido, tocando la melodía que has guardado. Empiezas en el listado 3.4 declarando y conectando un SqrOsc como la fuente de tu cadena de señal de audio (1). Luego defines algunas ganancias para encender y apagar tus notas (0.7 para encendido y 0.0 para apagado) (2). Luego declaras e inicializas un arreglo tal como lo hiciste antes, pero esta vez tienes más datos (¡más notas para tu canción!) (3).

Es hora de un nuevo método: a.cap() (cap por capacity, capacidad), que retorna el tamaño del arreglo a[]. En este caso el tamaño es 14. Puedes

entonces usar este número en un bucle for para iterar sobre cada elemento en el arreglo a[]. Imprimes el valor del índice i y los contenidos del arreglo en ese índice a[i] (4). Observa que cuando ejecutas el código, se corresponde con el dibujo del arreglo de la figura 3.1.

Todo calza musicalmente cuando tu nota MIDI del arreglo a[i] es convertida en frecuencia con el método Std.mtof() (5). Después de eso, es el método standard de tocar notas: definir la ganancia a un número distinto de cero (6), avanzar el tiempo (7), y apagar la nota por un momento (8) antes de volver y tocar la siguiente nota. Esto termina cuando i llega a 14 y no quedan más notas que tocar.

Listado 3.4 Tocar una melodía almacenada en un arreglo

```
//usemos un oscilador de onda cuadrada
//(1) oscilador de onda cuadrada para la melodía
SqrOsc s => dac;

//ganancias para separar nuestras notas
//(2) ganancias de notas encendidas y apgadas
0.7 => float onGain;
0.0 => float offGain;

//declarar e inicailzar un arreglo de notas MIDI
//(3) arreglo de notas MIDI para la melodía
[57, 57, 64, 64, 66, 66, 64, 62, 62, 61, 61, 59, 59, 57] @=> i

//bucle que recorre cada element del arreglo
for (0 => int i; i < a.cap(); i++) {
  //(4) imprime el índice y la nota del arreglo
  <<< i, a[i] >>>;

  //define la frecuencia y la ganancia para prender tu nota
```

```
    //(5) define la altura de las notas de melodía
    Std.mtof(a[i]) => s.freq;
    //(6) encendido de la nota
    onGain => s.gain;
    //duración de la nota encendida
    0.3 :: second => now;;

    //apaga tu nota para separarla de la siguiente
    //(8) nota apagada
    offGain => s.gain;
    0.2 :: second => now;
}
```

3.4 Grabar otros tipos de datos en un arreglo

Puede que hayas notado que la versión de "Twinkle" producida por el listado 3.4 está casi completa, a excepción de algunos problemas con el tiempo. Por ejemplo, las notas correspondientes a "star" y "are" deberían ser más largas que las otras. A continuación, aprenderemos que podemos almacenar prácticamente cualquier tipo de datos en arreglos, incluyendo tipo int, float, string e incluso duration.

3.4.1 Uso de un arreglo para almacenar datos tipo duration

La melodía "Twinkle" requiere que algunas notas sean más largas que otras. Los que escriben y leen música le llaman notas negras a las duraciones de las secciones "little" y a las duraciones más largas "star",

notas blancas. Estas son mostradas en notación musical en la figura 3.2.

Las notas negras corresponden a un cuarto de la duración de una nota redonda, mientras que las notas blancas corresponden a dos negras, y por lo tanto, a la mitad de una redonda.

Para tomar en cuenta esto en el código, puedes cambiar tu programa para que use lógica, con tal de sostener estas dos notas durante más tiempo. Si reemplazas la línea (7) del listado 3.4 con la declaración if/else del listado 3.5, entonces las dos notas serán sostenidas por el doble de tiempo. El tiempo de noteOff de 0.2 segundos es el mismo en todos los casos. Esta vez, aparte de un tiempo de noteOn de 0.8 segundos (1), hay notas blancas de un segundo. Y el tiempo de noteOn de 0.3 segundos, sumado a los 0.2 segundos de noteOff resulta en en los deseados 0.5 segundos de las notas negras (2). ¡Pruébalo y escucharás la diferencia!

Listado 3.5 Nueva lógica para controlar duraciones de notas

```
if (i == 6 || i == 13)
{
  //(1) algunas notas son más largas
  0.8 :: second => now;
}
else
{
  //(2) el resto son más cortas
  0.3 :: second => now;
}
```

También hemos dicho que los arreglos pueden almacenar cualquier tipo de datos, lo que es realmente poderoso para hacer música o cualquier tipo de

programación en general. Si piensas en todas las cosas que te gustarían manipular para hacer una canción, podrías tener en tu lista altura, duración, volumen entre otros (¡incluso letras!). Podrías guardar cualquiera de estos o todos en arreglos.

Almacenemos tus duraciones de notas en un arreglo, como se muestra en el listado 3.6. Aquí realizas la misma configuración del listado 3.4. Primero declaras y conectas el oscilador de onda cuadrada (1). Luego declaras e inicializas las ganancias para noteOn y noteOff (2) y creas tu arreglo de notas MIDI (3) (ahora se lama myNote, según tu aprendizaje de que las variables tienen que tener nombres con sentido). Declaras dos variables de tipo duración, una para las notas negras llamada q (de quarter note, su nombre en inglés) (4) y una para las notas blancas llamada h (de half note, su nombre en inglés) (5). Luego usas esos nombres cortos de variables para declarar e inicializar un arreglo de duraciones llamado myDurs (6).

Una vez que todo ha sido declarado, puedes introducir un bucle for para tocar tu canción, accediendo al arreglo de notas de la misma manera (8), encendiendo cada nota usando la variable onGain que declaraste (9). Pero ahora puedes usar tu arreglo myDurs para tocar cada nota según la duración correcta mediante hacer ChucKing de la duración apropiada a now (10). Después de que el tiempo pasa, apagas cada nota (11) y haces ChucK de tu duración de apagado a now. Repites para cada nota; luego el bucle termina cuando tu variable contador i alcanza el tamaño de tu arreglo de notas myNotes.cap() (7).

Listado 3.6 Almacenammiento de duraciones en un arreglo

```
//Hagamos Twinkle con una onda cuadrada
//(1) Oscilador de onda cuadrada para melodía
```

```
SqrOsc s => dac;

//ganancias para separar nuestras notas
//(2) Ganancias de encendido y apagado
0.7 => float onGain;
0.0 => float offGain;

//declarar e inicializar arreglo de notas MIDI
//(3) Arreglo de notas MIDI (int) para melodía
[57, 57, 64, 64, 66, 66, 64, 62, 62, 61, 61, 59, 59, 57] @=> i

//duraciones de notas negras y blancas
//(4) Duración de notas negras
0.3 :: second => dur q;
//(5) Duración de notas blancas
0.8 :: second => dur h;
//(6) Arreglo de duraciones para notas de la melodía
[q, q, q, q, q, q, h, q, q, q, q, q, q, h] @=> dur myDurs[];

//bucle de duración del arreglo
//(7) El bucle for itera sobre el arreglo de notas
for (0 => int i; i < myNotes.cap(); i++) {
  //(8) Define la altura de las notas de la melodía
  Std.mtof(myNotes[i]) => s.freq;
  //(9) Enciende la nota
  onGain => s.gain;
  //(10) Trasncurre el tiempo según la duración en el arreglo
  myDurs[i] => now;

  //apaga nuestra nota para separarla de la siguiente
  //(11) Apaga la nota
  offGain => s.gain;
  0.2 :: second => now;
}
```

3.4.2 Arreglos de strings: el texto también puede ser musical

Ahora que ya sabes almacenar cosas en arreglos, pongamos algunas palabras en un arreglo de strings. El listado 3.7 te muestra cómo hacerlo. Tal como tus otros arreglos, usa los mismos paréntesis cuadrados [...] par almacenar tu lista de strings (1). Separa elemento con comas y usa el operador ChucK @=> de copia (2) para almacenar esa lisat en un arreglo recién declarado de tipo string, el que se llamará words[] (3). Observa aquí que cada elemento tiene un largo diferente como string individual ("Twin" versus "kle"), y que ChucK se encarga de hacer todo por ti, reclamando el espacio de almacenamiento justo para contener lo que has declarado.

Extendiendo el código a través de varias líneas

> Observa aquí que hemos usado una característica de programación en ChucK que permite que una línea de código pueda ser extendida a través de varías líneas de texto. El punto y coma es lo que finaliza una línea de código ejecutable de ChucK, así que puedes escribir la mitad de tu lista de palabras, notas o lo que sea en un línea y la otra mitad en la siguiente (o incluso extender a más líneas), y ChucK se encarga de juntar todo cuando encuentra el primer punto y coma.

Listado 3.7 Un arreglo de strings (la letra de "Twinkle")

```
//haz un arrelgo para contener todas las letras y sílabas
//(1) Declara e inicializa un arreglo de strings para las letra
["Twin", "kle", "twin", "kle", "lit", "tle", "star",
//(2) Usa la forma del operador ChucK de copia @=>
"how", ""I, "won", "der", "what", "you", "are." ] @=> string wo
//(3) ChucK se encarga de calcular el tamaño del arreglo words
```

3.5 Ejemplo: una canción con melodía, armonía y letras

Para finalizar este capítulo, queremos darte un ejemplo más completo de cómo hacer una composición completa usando todo lo que hemos visto hasta el momento. Harás una versión mucho más larga de "Twinkle", usando dos arreglos de notas MIDI para melodía y armonía. Usarás un arreglo float para almacenar las duraciones de las notas. ¡También usarás un arreglo string para imprimir las letras de la canción a medida que se reproduce! Y harás paneo aleatorio de la melodía, usando la biblioteca Math. ¡Manos a la obra!

En el listado 3.8, primero declaras y añades paneo a tu oscilador de melodía usando el objeto Pan2 (1). A continuación, haces otro oscilador para armonía y lo conectas al dac (2). como has aprendido, el dac se hace cargo de mezclar estas fuentes sonoras automáticamente. Luego crea unas variables gain para encender y apagar tus notas (3).

A continuación, tienes el mismo arreglo de melodía con el que has trabajado (4), y haces otro arreglo para controlar el oscilador de armonía (5). Haz un arreglo dur que usarás para las duraciones (6) y otro arreglo string que contiene las palabras (7).

Ten cuidado de hacer todos estos arreglos del mismo largo, ya que si no podrías encontrar errores cuando pidas el elemento [i-ésimo] si no existe porque uno de los arreglos es más corto que los otros.

Continua con el mismo bucle for del último ejemplo usando la función .cap() para determinar el tamaño de los arreglos que estás usando (8). En el bloque del bucle, imprime el valor de tu variable contador i, en conjunto con los números de nota MIDI de melodía y armonía y, lo más importante, tu palabra o fragmento d epalabra correspondiente a esa nota en particular de la canción (9).

Luego define las frecuencias de ambos osciladores (10), pero añade variedad para la línea melódica, usando un valor aleatorio de paneo para cada nota (11). Enciendo tus dos osciladores por medio de definir sus ganancias como la variable ya definida onGain (12); luego avanza el tiempo por medio de la lectura y uso de los valores en el arreglo durs[] (13), (14)..

NOTA En el listado 3.8 usas un método de atajo (12) para configurar los valores de múltiples cosas a un valor común único. onGain => t.gain => s.gain; funciona bien, porque primero se hace ChucKing de onGain a t.gain, el que luego tiene esta valor, que se la hace después ChucKing a s.gain.

Listado 3.8 ¡"Twinkle" con melodía, armonía y letras!

```
//por el equipo ChucK, julio 2050

//dos osciladores, melodía y armoní
//(1) SinOsc a través de Pan2 para la melodía
SinOsc s => Pan2 mpan => dac;
//(2) TriOsc en el centro para la armonía
TriOsc t => dac;

//usaremos estos para separar entre notas
//(3) ganancias para encendido y apagado
0.5 => float onGain;
```

```
0.0 => float offGain;

//declara e inicializa los arreglos de números de notas MIDI
//(4) arreglo de enteros para notas MIDI de melodía
[57, 57, 64, 64, 66, 66, 64.
62, 62, 61, 61, 59, 59, 57] @=> int melNotes[];
//(5) arreglo enteros para notas MIDI de armonía
[61, 61, 57, 61, 62, 62, 61,
59, 56, 57, 52, 52, 68, 69] @=> int harmNotes[];

//duraciones de notas negras y blancas
0.5 :: second => dur q;
1.0 :: second => dur h;
//(6) arreglo tipo duration
[q, q, q, q, q, q, h, q, q, q, q, q, q, h] @=> dur myDurs[];

//haz otro arreglo para las palabras
//(7) arreglo tipo string para las letras
["Twin", "kle", "twin", "kle", "lit", "tle", "star",
"how", ""I, "won", "der", "what", "you", "are." ] @=> string w

//itera sobre todos los arreglos
// (¡¡ asegúrate que tengan el mismo tamaño !!)
//(8) toca todas las notas en el arreglo
for (0 => int i; i < melNotes.cap(); i++)
{
  //imprime el índice, notas MIDI y letras de los arreglos
  //(9) imprime datos de notas, incluyendo letras
  <<< i, melNotes[i], harmNotes[i], words[i] >>> ;

  //define melodía y armonía a partir de los arreglos
  //(10) define la frecuencias a partir de los arreglos de not
  Std.mtof(melNotes[i]) => s.freq;
  Std.mtof(harmNotes[i]) => t.freq;

  //la melodía tiene un paneo aleatorio para cada nota
```

```
  //(11) paneo aleatorio para el oscilador de melodía
  Math.random2f(-1.0, 1.0) => mpan.pan;

  //las notas están encendidas durante el 70% de la duración d
  //(12) enciende ambos osciladores
  onGain => s.gain => t.gain;
  //(13) el 70% de de la duración del arreglo es tiempo encend
  0.7 * myDurs[i] => now;

  //el espacio entre notas es el 30% del arreglo duration
  //(13) el 30% de la duración del arreglo es tiempo apagado
  offGain => s.gain => t.gain;
  0.3 * myDurs[i] => now;
}
```

3.6 Resumen

Los arreglos pueden hacer la vida mucho m'as simple y más organizada para ti:

- Los arreglos pueden ser usados no solo para almacenar enteros; como melodías y números de notas, sino que también cualquier secuencia de números de punto flotante, como ganancias y frecuencias.
- Los arreglos pueden ser usados para almacenar datos tipo duration y string, incluyendo palabras, instrucciones e incluso nombres de archivo, o cualquier tipo de datos - unidades generadoras, ¡prácticamente lo que sea!.
- Puedes encontrar el tamaño de un arreglo usando el método .cap(), así que no tienes que hacer un recuento de los elementos cada vez que

quieras cambiar algo.

Ahora que has aprendido un poco sobre el lenguaje ChucK, podemos enfocar enuestra atención a hacer sonidos y música más ricos y realísticos. Nuestro siguiente capítulo trata de trabajo con archivos de audio (esencialmente arreglos que contienen sonido) y cómo los puedes usar y manipular para hacer aún más asombrosa tu música.