Bibliotecas: herramientas incluidas en ChucK

Este capitulo cubre:

- Funciones de bilbiotecas (métodos)
- La biblioteca standard de ChucK, conversión de unidades
- Conversión de tipo con la biblioteca standard
- La biblioteca matemática de ChucK
- Generación de números aleatorios con la biblioteca de matemáticas y función sinusoidal
- Uso de funciones de bibliotecas para hacer música más expresiva

Ahora que tienes a ChucK a tus pies y has aprendido los fundamentos de su funcionamiento, y has creado tu primer conjunto de composiciones, es hora de aprender cómo producir música y sonido de una forma aún más expresiva. También estarás aprendiendo cómo hacer que la programación sea más simple, además de que tus programas sean de más fácil lectura. En este capítulo introduciremos a las bibliotecas, que son conjuntos de herramientas incluidas en ChucK que te permitirán alcanzar estas metas.

En casi todos los lenguajes de programación existen algunos conjuntos generales de funciones útiles que son empaquetadas en el formato llamado biblioteca. Estas bibliotecas se encargan de funciones básicas que los programadores tienen que usan en algún momento. Puedes pensar en estas funciones como herramientas, y estas herramientas son almacenadas en cajas de herramientas (cada biblioteca) de acuerdo a su tipo. Las herramientas individuales tomarán el nombre de métodos o

funciones, sobre lo que aprenderás más en el capítulo 4.

Revisaremos dos bibliotecas, o colecciones de funciones: la biblioteca ChucK Standard (llamada Std) y la biblioteca Chuck Math. Aprendiendo las funciones de la biblioteca Standard que convierten números y unidades, serás capaz de controlar altura y volumen mucho más simplemente. También aprenderás formas de convertir entre distintos tipos de datos, de strings a números y viceversa, por ejemplo. Con la biblioteca Math, aprenderás a generar y usar números aleatorios y cómo redondead números de punto flotante a enteros. También aprenderás algunos procedimientos matemáticos útiles, como calcular explícitamente valores de una función sinusoidal usando la función incluida sin() de la biblioteca Math, para controlar parámetros de síntesisde maneras musicales interesantes. Específicamente, en la sección 2.3 usarás la función sin() para suavizar el paneo de una señal hacia hacia izquierda y derecha en el campo sonoro stereo.

2.1 La biblioteca Standard: herramientas para altura, volumen y más

Para empezar, vamos a tratar de enfocarnos en los métodos de la biblioteca Standard que te permiten convertir unidades (como cambiar de grados Fahrenheit a centígrados, o los números que representan las notas en un teclado musical a las frecuencias para nuestros osciladores), y mostraremos cómo te pueden ayudar en tus composiciones.

¿Recuerdas del capítulo 1 que para tocar nuestra canción "Twinkle" tuviste que escribir muchos números para las alturas? Algunas de ellas, como 220 y 330, eran fáciles. ¿Pero cómo calculamos las otras? Otras, como

415.5305 Hz, podrían haberte llevado a preguntar "¿de dónde vino eso?". Bueno, una de las funciones más útiles de la biblioteca Standard te permite calcular la frecuencia de cualquier nota musical. Además, la biblioteca Standard tiene muchas otras funciones útiles, como una que te permite calcular lo relacionado a tu percepción de volumen, y otras funciones para convertir entre distintos tipos de datos. Armado de un par de funciones de la biblioteca Standard, serás capaz de tocar nuesra canción "Twinkle" mucho más fácilmente, y tu código será mucho más amigable y de fáci llectura. ¡Exploremos entonces algunas de las funciones de la biblioteca Standard!

La Interfaz Digital de Instrumentos Musicales (MIDI, por la sigla en inglés)

> La Interfaz Digital de Instrumentos Musicales (MIDI) fue adoptada en los años 1980s por casi todos los fabricantes de sintetizadores de teclados y electrónicos, órganos y pianos digitales y luego por los fabricantes de copmputadores, entre otros. MIDI le permitió a los instrumentos musicales electrónicos, controladores, computadores, programas de computador de notación musical y más, comunicar entre ellos cosas musicales, usando el mismo conjunto de códigos y números preestablecidos. Por ejemplo, un computador puede ordenarle a un sintetizador que toque una nota mediante el envío del mensaje NoteOn o alterar al altura enviando un mensaje PitchBend.

2.1.1 Derivando frecuencias musicales de números de notas MIDI

El primer método que vas a aprender es el conversor de nota MIDI a frecuencia, conocido en ChucK como Std.mtof() (mtof, por MIDI to

frequency, de MIDI a frecuencia). Así como con los nombres de las variables, verás que las funciones de la biblioteca casi siempre tienen nombres con sentido, aunque puedan parecer crípticos al principio.

Para convertir de números de notas MIDI a a frecuencias, primero tienes que saber cómo funcionan los números de notas MIDI. Los números de notas MIDI son enteros que corresponden a las teclas individuales de un un teclado musical, como se muestra en la figura 2.1. Las notas MIDI (teclas del teclado) son designadas por números enteros en el rango entre 0 y 127 (lo que corresponde al standard MIDi para la mayor parte de los parámetros MIDI). Los números de notas MIDI son dispuestos con tal que el C central (también llamado C) sea cercano a 60, cerca de la mitad del rango de notas MIDI.

El teclado musical standard tipo pianos

> Este tipo de teclado de teclas blancas y negras es denominado frecuentemente como teclado de piano o teclado musical. Esto es útil para distinguirlo de otros teclados, como el teclado QWERTY del computador en el que escribes.
>
> Algunos fabricantes decidieron que el C central debería ser C3, lo que corresponde a la nota MIDI #48, así que podrías encontrar diferencias en octavas entre distintos dispositivos y software computacional. Nosotros vamos a seguir con el C central correcto según la teoría musical, esto es, C central = C4 = 60.

Los números de notas MIDI combinados con la función Std.mtof(), hacen fácil crear melodías, tocar escalas y hacer otras tareas musicaesl. Para probarla, escribe y corre este código:

```
<<< Std.mtof(57) >>>;
```

Deberías ver lo siguiente impreso en la ventana Console Monitor

```
220.000000 :(float)
```

¡Corresponde la frecuencia de tu primra nota "Twinkle"!

FUNCIONES: ARGUMENTOS Y VALORES DE RETORNO

El número en los paréntesis (57 en este caso) es llamado el argumento de la función. La ejecuión de una función recibe el nombre de invocar o llamar a la función. El valor que resulta de usar la función es llamado el valor de retorno.

Ahora prueba escribiendo

```
<<< Std.mtof(60), Std.mtof(62), Std.mtof(64), Std.mtof(65), St
```

lo que debería arrojar lo siguiente en la ventana Console:

```
261.625565 293.664768 329.627557 349.228231 391.995436
```

Estas son las frecuencias de las primeras cinco notas de la escala de C: C, D, E, F, G. Revisando el teclado de la figura 2.1, puedes ver que si cuentas cada tecla (tanto blanca como negra), empezando en la 60 correspondiente a C4, las teclas blancas de la escala de C están asociadas a los números 60, 62, 64, 65 y 67.

Nota sobre programar en ChucK

> Tal como en el resto de la programación de computadores, existen múltiples maneras de lograr el mismo resultado. Las funciones de ChucK pueden ser usadas de dos maneras distintas: llamándolas con paréntesis y un argumento, como
>
> ```
> <<< Std.mtof(64) >>>;
> ```
>
> o haciendo ChucKing del argumento a la función:
>
> ```
> <<< 64 => Std.mtof >>>;
> ```
>
> Ambos hacen lo mismo y arrojarán el mismo resultado (valor de retorno).

Ahora, usemos un bucle for para tocar muchas notas usando la función Std.mtof. El código del listado 2.1 muestra un programa que toca la escala cromática completa (cada nota del teclado musical, tanto blanca como negra) usando un oscilador de onda triangular (TriOsc). Lo logras por medio de la creación de un bucle for que va de 0 a 127 (cubriendo cada nota MIDI), almacenando este valor en la variable i. Observa que la variable i se usa como entrada (también conocido como argumento de la función) a Std.mtof() (1). Entonces la primera vez que se recorre el bucle for, la nota MIDI 0 es convertida a 8.175799 Hz, que corresponde a una nota C que es muy grave como para ser escuchada, cinco octavas bajo el C central (también conocido como C-1).

Listado 2.1 Tocando una escala crómatica usando Std.mtof

```
//cadena de sonido
TriOsc t => dac;
0.4 => t.gain;

//bucle
for (0 => int i; i < 127; i++) {
  //(1) Usar Std.mtof para convertir de número de nota a frecu
  Std.mtof(i) => float Hz;
  //imprimir el resultado
  <<< i, Hz >>>
  //(2) Definir en Hz la frecuencia del oscilador
  Hz => t.freq;
  //Hacer avanzar el tiempo
  200 :: ms => now;
}
```

En la segunda iteración, la nota MIDI 1 es convertida a 8.661597 Hz. Cada vez que se ejecuta el bucle, el programa imprime el valor de la nota MIDI correspondiente y el valor convertido a frecuencia en Hz. El valor de frecuencia es ChucKed a t.freq (2) en cada iteración para que escuches la escala. En la tercera iteración, la nota MIDI 2 se convierte en 9.177024 Hz. No serás capaz de escuchar muchas de las primeras notas graves, porque son muy bajas en frecuencia como para ser posibles de reproducir con tus parlantes. Incluso cuando empieces a escuchar o sentir el sonido, tus oídos no serán capaces de percibir una altura en las frecuencias muy graves.

PRUEBA ESTO Corre el código del listado 2.1 varias veces, y trata de determinar la frecuencia más grave que puedes escuchar. Prueba con parlantes y con audífonos. ¿En qué frecuencia realmente empiezas a escuchar alturas en vez de solo sentir el sonido? El oído humano es más

sensible a ciertos rangos de frecuencias que otros. ¿Cuál es la frecuencia que escuchas a mayor volumen?

PRUEBA ESTO Reescribe el bucle for para tocar la escala cromática, usando un bucle tipo while. ¡Recuerda incrementar el contador dentro del bucle!

Como puedes imaginar, en adelante el método Std.mtof() va a jugar un gran rol en la creación de melodías. Es también posible ir en la otra dirección y usar el método Std.ftom() para convertir frecuencias en notas MIDI. Esto puede resultar útil si tú tienes alguna frecuencia particular y quieres saber la nota musical más cercana a ella, or si estás utilizando un sintetizador externo de software o hardware que acepta notas MIDI. Aprenderás sobre cómo enviar notas MIDI a sintetizadores y otros dispositivos conectados a tu computador en el capítulo 11.

NOTA PARA MÚSICOS Std.mtof() puede también aceptar valores de punto flotante para ayudarte con escalas microtonales y alturas, con lo que Std.mtof(60.5) resulta en un cuarto de tono entre C y C#.

2.1.2 Convirtiendo entre tipos de datos: float a int

El siguiente método de la bilbioteca Standard te permitirá convertir números de punto flotante en enteros. Cuando te empezaste a dar cuenta que necesitabas números de punto flotante para expresar de forma precisa frecuencias y ganancias, es posible que hayas escrito alguna línea que arrojara un error porque estabas tratando de almacenar una variable tipo float en otra tipo int. El ejemplo más simple de esto es

```
220.0 => int myFreq;
```

que arroja un error que es impreso en la ventana Console Monitor:

```
line(1): cannto resolve operator "=>" on types 'float' and 'in
```

Esto es ChucK protegiéndote de una forma, al decirte que podrías estar perdiendo en la conversión de número flotante a entero. Un ejemplo de esto sería

```
220.5 => int myInt;
```

donde la parte .5 sería desechada sin que te dieras cuenta. Es por esto que ChucK se niega a hacerlo e imprime un error. ChucK como lenguaje es de tipado fuerte, lo que significa que los tipos de datos son mantenidos con relativa pureza, y convertir de uno a otro puede requerir un poco de trabajo de tu parte. Interesantemente, convertir un int a un float no requiere trabajo, porque no se pierdan datos en el proceso. Entonces

```
220 => float myFloatFreq;
```

sí funciona, porque 220 es cambiado a 220.0, no se pierden datos, y de hecho se gana precisión, para uso posterior.

Ir en dirección contraria, de float a int, requiere lo que se llaama una conversión o casteo de tipo, lo que significa tomar datos de un tipo y castearlos en otro de la mejor forma posible en otro tipo de datos. La más

común que necesitarás como programador de ChucK es ir de float a int, lo que se logra usando la función Std.ftoi (float to integer, de punto flotante a entero) de la siguiente forma:

```
220.5 => Std.ftoi => int myFreq; (o también Std.ftoi(220.5) =>
```

Nota que en este caso, la parte fraccional es truncada (desechada), resultando en 220. Incluso si especificaras

```
Std.ftoi(220.999999999) => int myFreq;
```

también obtendrías 220, porque toda parte fraccional es desechada.

2.1.3 Obteniendo un entero a partir de un número expresado como texto

Otro ejemplo de conversión de tipo de datos podría ser si tuvieras una variable string con valor "128.7", y lo quisieras usar como float para definir una frecuencia u otra cosa. Muchos programs aceptan entradas del usuario mediante la escritura de la información en el computador, y ChucK también puede leer datos de tu computador, algunos de los que podrían contener texto. Afortunadamente, existen también métodos de la biblioteca Std para convertir strings (conocidos también como ASCII, de donde proviene la a de la función atoi) a y en enteros y números de punto flotante. Para hacerlo usa Std.atoi("128.7"); como se muestra en la tabla 2.1

Tabla 2.1 Funciones de la biblioteca Standard de ChucK para convertir datos de tipo string, float e int

Método	Resultado	Descripción
Std.atoi(string)	int	Convierte ASCII (string) en int
Std.atof(string)	float	Convierte ASCII (string) en float
Std.itoa(int)	string	Convierte int en ASCII (string)
Std.ftoa(float)	string	Convierte float en ASCII (string)

Existen más funciones útiles de la biblioteca Std, cuyo listado y descripción completos se encuentra en el apéndice B, incluyendo los de calcular el valor absoluto de números (la función valor absoluta retorna números negativos como positivos, y no altera los números positivos) y retorna el signo, negativo o positivo, de los números

Standard no es la única biblioteca disponible en ChucK, así que dejémosla por ahora y continuemos con otra biblioteca muy útil, llamada Math

2.2 La biblioteca Math de ChucK

¿Recuerdas cuán entretenido era cuando usabas Math.random() para definir las frecuencias y duraciones de notas en un bucle infinito? En las siguientes subsecciones, profundizaremos en otra biblioteca, conocida como Math, que tiene un gran número de funciones útiles que te permitirán hacer que tus programas sean mucho más poderosos y expresivos.

2.2.1 Funciones aleatorias de la biblioteca Math

El primer conjunto de métodos de la biblioteca Math es usado para generar

números aleatorios (random). Los usaste en el capítulo 1 para tocar un flujo infinito de frecuencias y duraciones aleatorias, así que ahora revisaremos cómo exactamente funcionan. Los números aleatorios son muy útiles para hacer arte digital. Es muy tedioso para un programador definir cada uno de los valores únicos de cada parámetro en el tiempo, al momento de explorar un sonido con muchos parámetros, por ejemplo. No obstante, el uso de funciones aleatorias te permite encargarles aquellas tareas al computador, mientras buscas las partes que te gustan, tal como lo hiciste con las notas aleatorias en el capítulo 1.

A pesar de que ChucK no es realmente un lenguaje de programación de juegos (aunque mucha gente ha escrito juegos con él), digamos que quieres simular tirar un dado. Necesitarás una manera de generar aleatoriamente un número que puede ser 1, 2, 3, 4, 5 o 6. En el ejemplo del listado 2.2, usarás la biblioteca Math para generar enteros aleatorios entre 1 y 6 usando el método Math.random2(). Este programa corre para siempre (la condición while(true) siempre es verdadera) y genera un número aleatoria cada medio segundo. El 2 en random2 significa que vas a especificar 2 números, un mínimo y un máximo, entre los que la función generará números aleatorios. Los números especificados también son permitidos, así que en este caso el 1 y el 6 aparecerán con igual probabilidad que todos los enteros entre 1 y 6.

Listado 2.2 Generación de números enteros usando la biblioteca Math

```
//generación de números enteros aleatorios
//simula tirar un dado
while(true) {
  <<< "el dado arroja = ", Math.random2(1,6) >>>;
  second / 2 => now;
}
```

La biblioteca Math tiene cuatro maneras de crear números aleatorios, como se ve en la tabla 2.2. Los enteros son hechos con los métodos Math.random2() y Math.random().

Tabla 2.2 Funciones de la biblioteca Math de ChucK para crear números aleatorios

Método	**Resultado**	**Descripción**
Math.random()	int	Genera enteros aleatorios entre 0 y Math.RAMDOM_MAx
Math.random2(int min, int max)	int	Genera enteros aleatorios en el rango [min, max]
Math.randomf()	float	Genera números de punto flotante en el rango [0, 1]
Math.random2f(float min, float max)	float	Genera números de punto flotante en el rango [min, max]

Los números aleatorios de punto flotante son creados usando Math.randomf(), que retorna un número entre 0.0 y 1.0, y Math.random2f(), que retorna un número dentro de un rango. Nuevamente, como programador tú tienes que especificar los valores mínimo y máximo.

NOTA: Math.random() no es usado frecuentemente, pero se incluye en ChucK para ser más compatible con otros lenguajes de programación.

El programa del listado 2.3 demuestra cóm los números aleatorios pueden ser utilizados en el proceso de composición. Como puedes ver, hemos definido un bucle for que toca un total de 16 notas (1). En cada iteración del bucle, un entero aleatorio entre 48 y 72es creato y almacenado en la

variable entera myNote (2), y un número aleatorio de punto flotante entre 0.05 y 0.9 es generado y almacenado en la variable myGain de tipo float (3). Después de imprimir ambos a la consola, myNote es usada para definir la frecuencia del oscilador s a través de tu viejo amigo Std.mtof() y myGain es usada para definir el volumen (4). Es así que una nueva nota MIDI (frecuencia) (5) y volumen son generados cada un quinto de segundo (6), haciendo una composición que genera notas aleatoriamente. Aunque no genera una melodía conocida, creemos que puedes estar de acuerdo en que es musical, debido a los cambios nota a nota de volumen (llamados acentos por músicos y compositores) y las siempre cambiantes frecuencias. Ejecútalo unas veces más y escucharás que las notas y los volúmenes generados son diferentes cada vez. Lo único que está garantizado es que serán exactamente 16 notas.

Listado 2.3 Música aleatoria usando la biblioteca Math

```
//Música aleatoria de onda cuadrada
SqrOsc s => dac;

//(1)El bucle for toca 16 notas
for (0 => int i; i < 16; i++) {
  //(2) Nota entera aleatoria (C3 – C5)
  Math.random2(48, 72) => int myNote;
  //(3) Ganancia aleatoria entre .05 y .9
  Math.random2f(0.05, 0.9) => float myGain;
  //(4) Imprime la nota y la ganancia actuales
  <<< myNote, myGain>>>;
  //(5) Define la frecuencia y la ganancia del oscilador
  Std.mtof(myNote) => s.freq;
  myGain => s.gain;
  //(6) Deja que cada note suene durante 1/5 de segundo
  0.2 :: second => now;
```

```
}
```

Los generadores de números aleatorios te permite escribir programas musicales que nunca sonarán exactamente iguales, pero puedes hacerlo de una manera bastante controlable, lo que puede ser una técnica atractiva pra un compositor moderno. Existe una manera de asegurarse que una secuencia de números aleatorios sea siempre la misma pero diferente de todas las otras secuencias. Imaginemos que que no quieres escribir cuáles son las notas específicas, y quieres que sean generadas aleatoriamente, pero quieres que tu composición sea siempre la misma cada vez que la ejecutas. ¿Cómo lograr esto?

La clave para controlar la generación de números aleatorias, además de especificar el rango, es generando una semilla, que el generador de números aleatorios sua para empezar la secuencia de números aleatorios que generará a partir de entonces. Típicamente la semilla es creada usando algo que está cambiando constantemente, como el reloj del sistema de tu computador. Entonces, cada vez que corres el programa del listado 2.3, una semilla diferente es generada y el programa genera un conjunto completamente nuevo de notas y volúmenes. Pero si empiezas con la misma semilla te aseguras que la secuencia de números aleatorios sea siempre la misma a lo largo de distintas ejecuciones del mismo programa. ¿Entonces cómo puedes definir tú mismo la semilla? Escribe la siguiente línea de código e insértala en el listado 2.3, justo antes del bucle for en (1):

```
//define la semilla (seed)
Math.srandom(134);
```

El número 134 en este ejemplo fue escogido de forma arbitraria. Prueba jugando con el valor de la semilla. Cámbialo a 133 y observa qué sucede. De forma contraria a lo que podrías pensar, cambiar la semilla solo un poco no genera cambios sutiles en la secuencia. Cada valor de semilla genera una secuencia completamente distinta, sin relación a ninguna otra.

PRUEBA ESTO Cambia el valor de la semilla en Math.srandom a unos pocos números distintos, y observa lo que le hace a tu composición. DEberías escuchar un patrón particular para cada valor de semilla pero distinto para cada semilla. Además, cambia los números en el rango especificado donde defines el número de nota ((2) en el listado 2.3). Por ejemplo, aumenta el número más alto, y observa que empiezas a escuchar ocasionalmente notas más altas que anteriormente.

2.2.2 Redondeando números: siendo más justos en las conversions float a int

Ahora has empezado a ver el poder de la biblioteca Math, y existen muchas más herramientas útiles en ella. ¿Recuerdas cuándo casteaste tus números float a enteros, y la parte fraccional fue perdida por el truncamiento? Recuerda que 220.9999 fue truncado a 220, lo que no parece justo. La solución es redondead, lo que significa que cualquier parte fraccional mayor o igual a 0.5 se redondea hacia arriba, y cualquiera menor se redondea hacia abajo (es truncada). Math provee una función llamada round que se encarga de esto. Pruébala con esta línea:

```
<<< Math.round(220.501), Math.round(220.49) >>>
```

Existen otras funciones de Math, incluyendo logaritmos, exponentes y

trigonometría (seno, coseno y tangente). Si estás familiarizado con estas funciones, puedes encontrar la lista en el ya familiar apéndice B. Estarás usando algunas de ellas en los siguientes capítulos a medida que avanzemos a programación avanzada y técnicas de procesamiento de señales.

2.3 Stereo y paneo

Tomemos un descanso de las matemáticas (pero solo uno breve, porque usaremos una función Math muy pront) para revisar uan técnica de audio entretenida conocida como paneo (panning). Paneo viene de la palabra panorama, lo que significa una "vista de una región que rodea a un observador". En audio, el paneo es una manera en que los mezcladores de audio ubican el sonido en el campo stereo entre los parlantes izquierdo y derecho. En la música moderna y la producción de sonido, muchas de las fuentes sonoras son grabadas en diferentes canales y a menudo en momentos y ubicaciones distintos. El mezclador (tanto el dispositivo llamado mezclador como la persona que lo opera) tiene que combinar los elementos de forma artificial para crear un sentido de espacio. Como se muestra en la figura 2.2, cuando se mezcla una canción, los ingenieros de audio a menudo ubican distintos elementos en regiones discretas (por ejemplo, los vientos a la izquierda y la guitarra más a la derecha).

Vamos a presentar tres métodos (mostrados en los listados, 2.4, 2.5 y 2.6) que puedes usar para hacer paneo en ChucK, y te otorgarán más parámetros para componer. El primer ejemplo muestra dos osciladores, uno siendo paneado fuertemente al parlante izquierdo y el otro al parlante derecho (o audífono izquierdo y derecho) usando dac.left (1) y dac.right (2) (left significa izquierda, right signfica derecha). ¡Exacto! el dac tiene algunas

funciones extra escondidas, entre ellas la conexión a distintos canales de audio. Cuando ejecutas este ejemplo, escucharás dos tonos sinusoidales que son distintos en altura (3), (4) en cada lado de tu campo stereo (una sinusoide a la izquierda y otra a la derecha).

Listado 2.4 Usando dac.left y dac.right para conectar a los parlantes izquierdo y derecho.

```
//dos ondas sinusoidales en stereo
//(1) conecta un SinOsc al canal izquierdo...
SinOsc s => dac.left;
//(2) ... y otro al canal derecho
SinOsc t => dac.right;

//definir frecuencias
//(3) define la frecuencia del oscilador izquierdo...
220 => s.freq;
//(4) ... y la frecuencia del derecho a una distinta
361 => t.freq;

//avanzar el tiempo
second => now;
```

La segunda técnica para panear es similar, pero puede ser usada para hacer audio multicanal (como el sistema 5.1 envolvente de sonido en los cines y usados en muchos sistemas de cine casero y videojuegos). Si tienes una configuración en tu estudio con más de dos parlantes, y tienes una tarjeta multicanal de sonido o una interfaz externa que soporta más de dos canales, entonces puedes usar el método dac.chan() (chan por channel, que significa canal) para definir qué quieres reproducir en cada parlante. El siguiente listado muestra la conexión multicanal de cuatro osciladores.

Listado 2.5 Usando dac.chan() para conectar a múltiples parlantes

```
SinOsc s0 => dac.chan(0);
SinOsc s1 => dac.chan(1);
SinOsc s2 => dac.chan(2);
SinOsc s3 => dac.chan(3);
```

El tercer método de paneo es el que más nos gusta, porque nos permite usar nuestras habilidades de programación y el poder de ChucK para lograr cosas que generalmente no podemos en programas de audio comerciales.

Existe un entetenido objeto UGen llamado Pan2, que puedes insertar en la cadena de audio (entre tu fuente SinOsc y el dac), como se muestra en la figura 2.6. Hemos nombrado este objeto p, y lo puedes conectar al dac (salida de audio) (1). Lo bueno de conectar Pan2 al dac es que ChucK sabe que son stereo, por lo que las salidas correspondientes de Pan2 son automáticamente conectadas a dac.left y dac.right. El paneo del objeto Pan2 es controlado con un número de punto flotante entre -1.0 (izquierda) y 1.0 (derecha), usando el método .pan. Como puedes ver en el siguiente listado, defines la variable panPosition como -1.0 (izquierda) (2). La posición de paneo es definida según la actualización de p.pan con el valor actual de panPosition (4). Aumentas lentamente la variable (hacia la derecha) en incrementos de .01 (5) hasta que llega a 1.0 (3). Puedes escuchar mejor este ejemplo con audífonos.

Listado 2.6 Usando un objeto Pan2 para conectar un SinOsc a la salida dac stereo

```
//ejemplo de paneo
```

```
//(1) Corre el oscilador a través del objeto Pan2
SinOsc s => Pan2 p => dac;

//inicializar la posición de paneo
//(2) define a la izquierda el paneo inicial
-1.0 => float panPosition;

//hacer un bucle para variar el paneo
//(3) hasta que el paneo llegue a la derecha
while (panPosition < 1.0) {
  //(4) define la nueva posición de paneo...
  panPosition => p.pan;
  <<< panPosition >>>;
  //(5) ... y la incrementa un poco
  panPosition + 0.01 => panPosition;
  10 :: ms => now;
}
```

Para hacer este paneo aún más interesante, vamos a volver a la biblioteca Math y mirar otro ejemplo usando el objeto Pan2 con un nuevo objeto gnerador de sonido llamado Noise (ruido). El objeto Noise genera rudido blanco (valores aleatorios en cada punto en el tiempo e igual energía para cada frecuencia), que se aproxima mucho al sonido "sss" de "see" de la figura 1.1.

Entonces, conecta la fuente Noise n al objeto de paneo Pan2 y al dac. En un bucle wihle infinito, usa un nuevo método llamado Math.sin() (seno). Este método usa una función seno para crear nuevos parámetros para el objeto de paneo. Observa cómo esto suena. El ruido blanco se mueve entre la izquierda y la derecha, adelante y atrás, en el patrón de una onda sinusoidal, para siempre. Esto puede ser una herramienta poderosa para hacer composiciones expresivas.

Listado 2.7 Paneo automático usando pan2 y la función Math.sin()

```
//cadena de sonido; ruido blanco a pan2 a dac
Noise n => Pan2 p => dac;

//el ruido puede sonar demasiado fuerte
0.2 => n.gain;

//bucle infinito
while (true) {
  //oscila el paneo entre -1.0 y 1.0
  Math.sin(now/second) => p.pan;
  //hazlo muy a menudo, para hacerlo más suave
  ms => now;
}
```

Existen muchas otras funciones matemáticas basadas en trigonometría en la biblioteca Math, las que están listadas junto a a todas las otras funciones de las bibliotecas Std y Math en el apéndice B. ¿Cuál es la diferencia entre Math.sin() y el objeto SinOsc? El objeto SinOsc es un objeto que emite sonido y de tipo UGen. Como tal, SinOsc automáticamente produce una onda sinusoidal al ser conectada al dac, pero Math.sin() requiere que la llames con un argumento, y luego retorna un punto específico dentro de una onda sinusoidal. Las funciones de la biblioteca Math sirven para hacer cálculos matemáticos y también como parámetros, como controlar métodos .pan o .gain. Los usos compositivos de las funciones Math pueden incluir la añadidura de vibrato o incluso la generación de notas.

Ahora que has adquirido muchas nuevas herramientas, las funcioens de las bibliotecas Std y Math, deberías usarlas para hacer música. Así que hagamos justo eso.

2.4 Ejemplo: música aleatoria con dos voces y paneo

Para cerrar este capítulo, harás un generador de música aleatoria usando muchas de los atractivos métodos de las bibliotecas que hemos visto. En este ejemplo, tendrás una voz solista, un oscilador tocando arriba y abajo del teclado aleatoriamente, y tendrás una voz de acompañamiento que se queda en las notas que la voz solista toca, pero una octava más grave. Además, harás paneo aleatorio de de la melodía, usando la biblioteca Math. ¡Manos a la obra!

Como se muestra en el listado 2.8, primero declara y añade paneo a tu oscilador de melodía usando un objeto Pan2 (1). A continuación, haz otro oscilador para armonía y conéctalo al dac (2). Por supuesto, el dac se encarga de mezclar estas fuentes automáticamente.

Después crea variables para prender y apagar tus notas (3). Como último paso de configuración, declara un número de nota MIDI para que tu melodía empiece en (4).

Listado 2.8 Caminata aleatoria a dos voces con paneo

```
//Caminata aleatoria a dos voces con paneo
//Por el equipo ChucK, 25 de septiembre, 2020

//dos osciladores, melodía y armonía
//(1) SinOsc a través de Pan2 para poder otorgarle movimiento
SinOsc s => Pan2 mpan => dac;
//(2) TriOsc en una posición fija al centro
TriOsc t => dac;
```

```
//usaremos estos más adelante para separar las notas
//(3) variables tipo float pra controlar tus ganancias de notas
0.5 => t.gain;
0.5 => float onGain;
0.5 => float offGain;

//variable entera para controlar tu melodía
72 => int melodyNote

while (true) {

  //define la altura de la melodía aleatoriamente, con límites
  //(5) cambia la melodía aleatoriamente hacia arriba o hacia a
  Math.random2(-3, 3) +=> melodyNote;

  //(6) límite inferior de la melodía
  if (melodyNote < 60) {
    60 => melodyNote
  }
  //(7) límite superior de la melodía
  if (melodyNote > 84) {
    84 => melodyNote;
  }

  //(8) define la altura del oscilador SinOsc
  Std.mtof(melodyNote) => s.freq;

  //"la melodía tiene un paneo aleatorio para cada nota
  Math.random2f(-1.0, 1.0) => mpan.pan;

  //"tira un dado", cambia la nota de la armonía
  if (Math.random2(1,6) == 1) {
    //(9) mantiene aleatoriamente la altura del TriOsc
    Std.mtof(melodyNote - 12) => t.freq;
  }
```

```
    //elige aleatoriamente entre tres duraciones
    Math.random2(1.3) * 0.2 => float myDur;

    //la nota suena durante el 80% de la duración
    onGain => s.gain;
    (myDur * 0.8) :: second => now;

    //el espacio entre notas es el 20% de la duración
    offGain => s.gain;
    (myDur * 0.2) :: second => now;
}
```

En el bloque while de bucle infinito, estás aleatoriamente aumentando o disminuyendo tu nota de melodía MIDI nota entre -3 y +3 ntoas (teclas del teclado MIDI) (5). Esto recibe el nombre de caminata aleatoria, donde en vez de definir la nota en sí misma como un número aleatorio, caminas aleatoriamente hacia arriba o abajo del teclado (o te mantienes quieto si es que el número aleatorio es 0). Para asegurarte que no caminas muy a la izquierda o muy a la derecha, tocando así notas muy altas o bajas en altura, compruebas la nueva nota MIDI para asegurarte que esté entre C central (6) y dos octavas más arriba (7). Una vez que has definido tu nueva nota, usas Std.mtof para convertir el número en una altura para tu SinOsc (8). También defines un nuevo paneo aleatorio para tu melodía (a través de Pan2) entre izquierda (-1.0) y derecha (1.0).

A continuación, arrojas un dado (números entre 1 y 6 incluidos) y decides si cambiarás la altura de tu armonía TriOsc (9). Existe una posibilidad uno a seis de hacerlo. Si cambia, se le da la misma altura que el oscilador de melodía pero una octava más grave (+/- 12 en notas MIDI es subir o bajar una octava).

Finalmente, calculas una duración aleatoria entre 0.2, 0.4 y 0.6 segundos (1, 2 o 3 multiplicado por 0.2), para tu nota de melodía hasta que corra de nuevo el bucle. Enciendes la melodía durante el 80% del tiempo y lo apagas durante el 20%. Luego el bucle se ejecuta de nuevo, para siempre, hasta que hagas click en Remove Shred o en Remove All Shreds.

PRUEBA ESTO Añade una llamada a Math.srandom() para definir la semilla de generación de números aleatorios, antes del bucle while principal. Cambia el valor de la semilla un par de veces, observando que cada número genera una canción distinta pero igual si usas la misma semilla. Encuentra una semila que genere tu nueva y original canción favorita.

2.5 Resumen

¿Vaya, no? Acabas de hacer una composición musical sonora interesante, pero optaste por dejarle la mayoría de las decisiones reales sobre notas y duraciones y sincronización de la melodía y de la armonía completamente a cargo del computador. Fuiste capaz de esto usando todo lo que has aprendido hasta el momento:

- Uso de funciones de la biblioteca Standard como Std.mtof
- Conversión de variables a y float (y al revés) y redondeo
- Uso de funciones de la biblioteca Math como Math.random
- Paneo usando el UGen Pan2
- Composición de música aleatoria y control de la aleatoreidad (rango y semilla)

Harás mucho más con las funciones de las bibliotecas en el futuro, y aprenderás cómo escribir tus propias funciones desde cero (capítulo 5).

El siguiente capítulo tratará sobre arreglos, que son colecciones de datos que puedes usar para organizar tus notas, alturas, duraciones, parámetros, incluso strings de texto, o lo que sea. Una vez que sepas cómo crearlos y usarlos, serás capaz de programar canciones completas más fácilmente, escribiendo mucho menos, y también harás que tus programas sean de más fácil lectura para ti y otros usuarios.