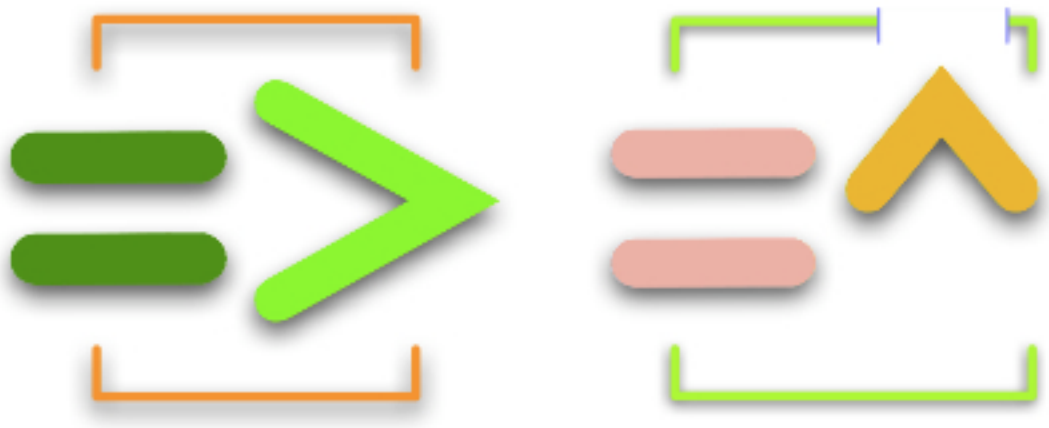
ERICA

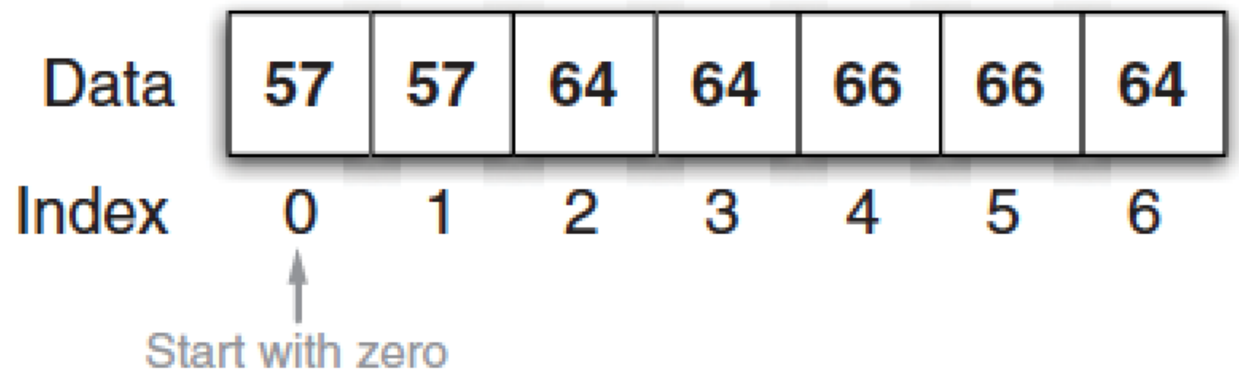
소프트웨어학부

CSE2020 음악프로그래밍



3
Arrays:
arranging and accessing your compositional data

Declaring and storing data in arrays



Listing 3.1 Declaring and filling an array of integers the long way

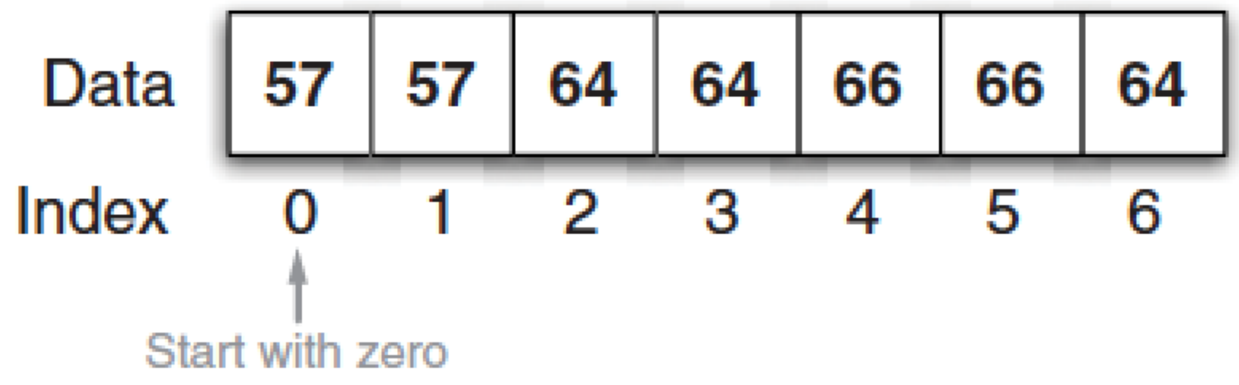
```
// array declaration (method 1)
int a[7];

57 => a[0];
57 => a[1];
64 => a[2];
64 => a[3];
66 => a[4];
66 => a[5];
64 => a[6];

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

1. Declares an array of specific length (7). (7).
2. Sets value stored in the zeroeth element…

…and all locations…

3. …up to and including the last element.

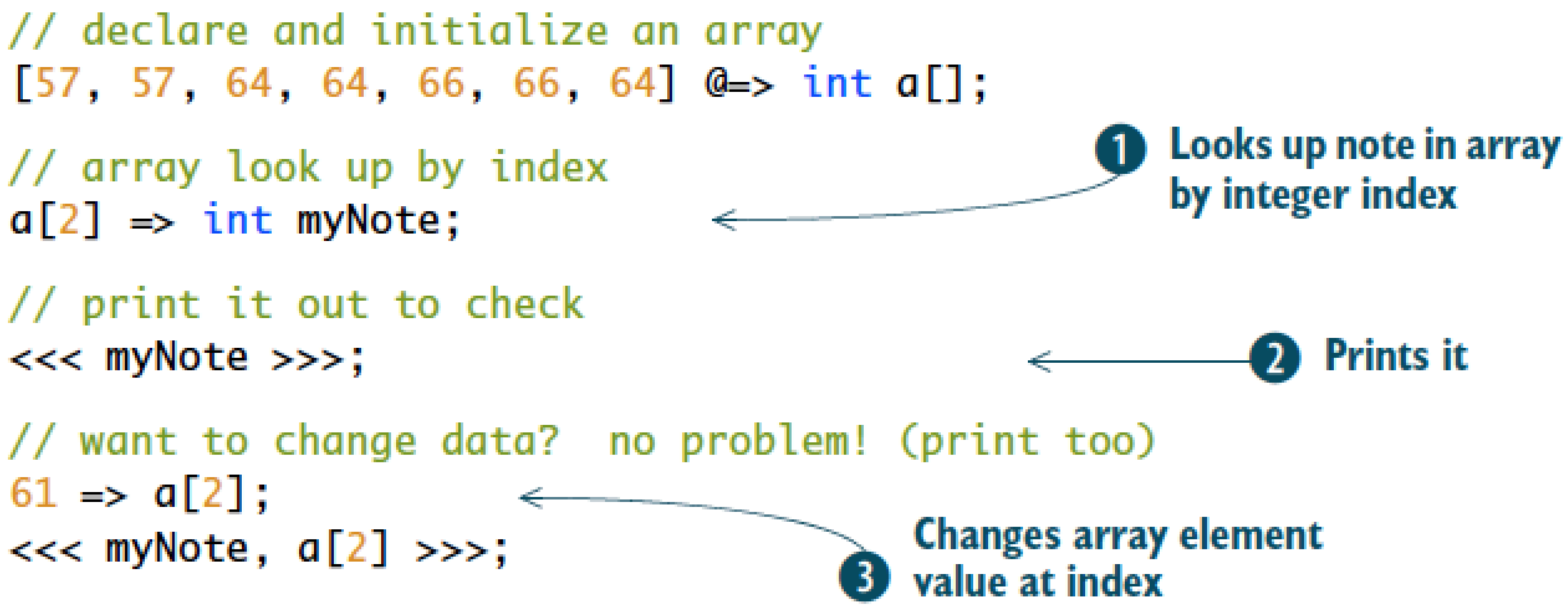
Declaring and storing data in arrays



Listing 3.2 Declare and initialize an array all at once

```
[57, 57, 64, 64, 66, 66, 64] @=> int a[];
<<< a[0], a[1], a[2], a[3], a[4], a[5], a[6] >>>;
```

Reading and modifying array data



```
// declare and initialize an array
[57, 57, 64, 64, 66, 66, 64] @=> int a[];

// array look up by index
a[2] => int myNote;

// print it out to check
<<< myNote >>>;

// want to change data?  no problem! (print too)
61 => a[2];
<<< myNote, a[2] >>>;
```

1 Looks up note in array by integer index

2 Prints it

3 Changes array element value at index

Adding and removing elements in array

```
[64, 65, 60, 59] @=> int notes[];
notes << 58 << 60; // notes is now [64, 65, 60, 59, 58, 60]
```

Table B.11 Array-supported functions

Function name and arguments	What it does
`int size();`	Number of elements in the array.
`int size(int n);`	Sets number of elements in the array to n. Fills any new elements with default values (`0` for `int` and `float`, `0::second` for `dur` and `time`, `null` for strings and objects).
`void popBack();`	Removes last element in array.
`void clear();`	Removes all elements in array.

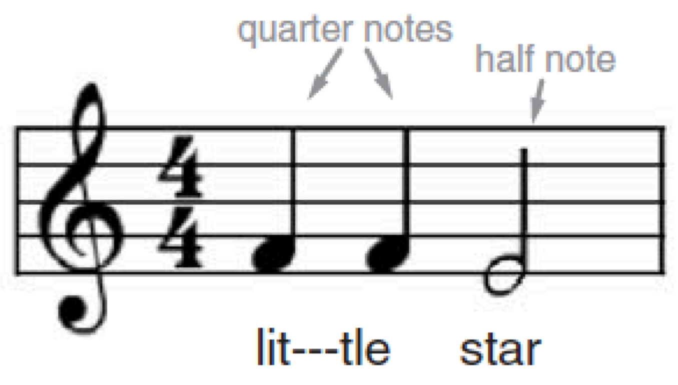
Using array data to play a melody

Listing 3.4 Playing a melody stored in an array

```
//  Let's Twinkle with a square wave
SqrOsc s => dac;

// gains to separate our notes
0.7 => float onGain;
0.0 => float offGain;

// declare and initialize array of MIDI notes
[57,57,64,64,66,66,64,62,62,61,61,59,59,57] @=> int a[];

// loop for length of array
for (0 => int i; i < a.cap(); i++)
{
    <<< i, a[i] >>>;

// set frequency and gain to turn on our note
    Std.mtof(a[i]) => s.freq;
    onGain => s.gain;
    0.3::second => now;

// turn off our note to separate from the next
    offGain => s.gain;
    0.2::second => now;
}
```

1. Square wave oscillator for melody
2. Note on/off gains
3. Array of MIDI notes (int) for melody
4. Prints index and array note
5. Sets pitch for melody notes
6. Note on
7. Duration for note on
8. Note off

Controlling note durations



Listing 3.5 New logic to control note durations

```
if (i==6 || i==13)
{
    0.8::second => now;        ← 1 Some notes are longer
}
else
{
    0.3::second => now;        ← 2 The rest are shorter
}
```

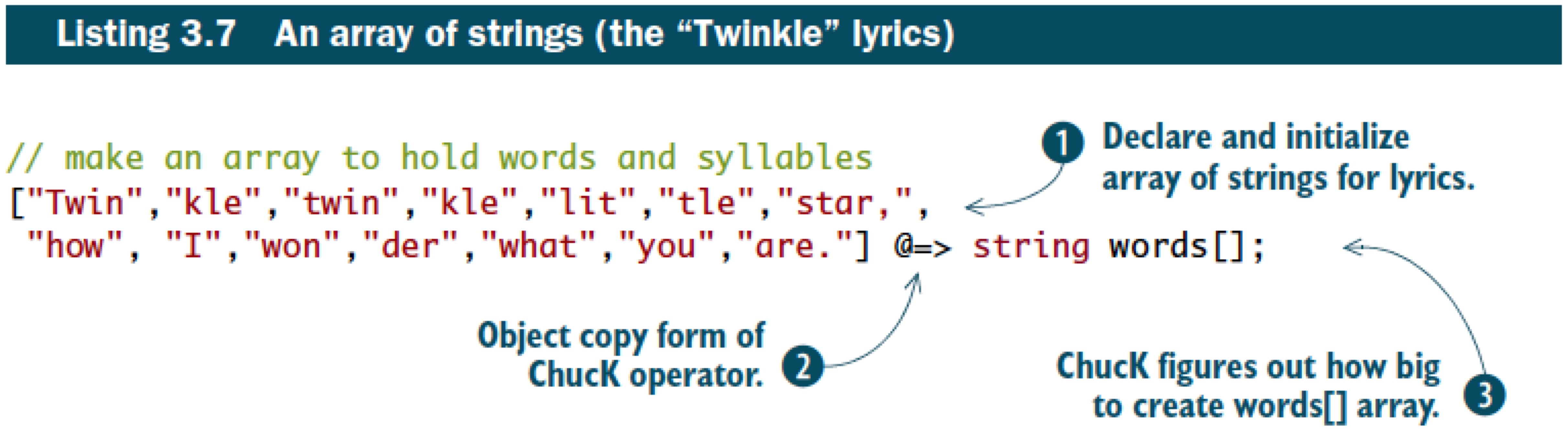
Using an array to store durations

Listing 3.6 Storing durations in an array

```
//  Let's Twinkle with a square wave
SqrOsc s => dac;

// gains to separate our notes
0.7 => float onGain;
0.0 => float offGain;

// declare and initialize array of MIDI notes
[57,57,64,64,66,66,64,62,62,61,61,59,59,57] @=> int myNotes[];

// quarter note and half note durations
0.3 :: second => dur q;
0.8 :: second => dur h;
[q, q, q, q, q, q, h, q, q, q, q, q, q, h] @=> dur myDurs[];

// loop for length of array
for (0 => int i; i < myNotes.cap(); i++)
{

    // set frequency and gain to turn on our note
    Std.mtof(myNotes[i]) => s.freq;
    onGain => s.gain;
    myDurs[i] => now;

    // turn off our note to separate from the next
    offGain => s.gain;
    0.2::second => now;
}
```

1. Square wave oscillator for melody
2. Note on/off gains
3. Array of MIDI notes (int) for melody
4. Duration for quarter notes
5. Duration for half notes
6. Array of durations for melody notes
7. For loop iterates over length of note array
8. Sets pitch for melody notes
9. Note on
10. For duration stored in array for that note
11. Note off

Arrays of strings

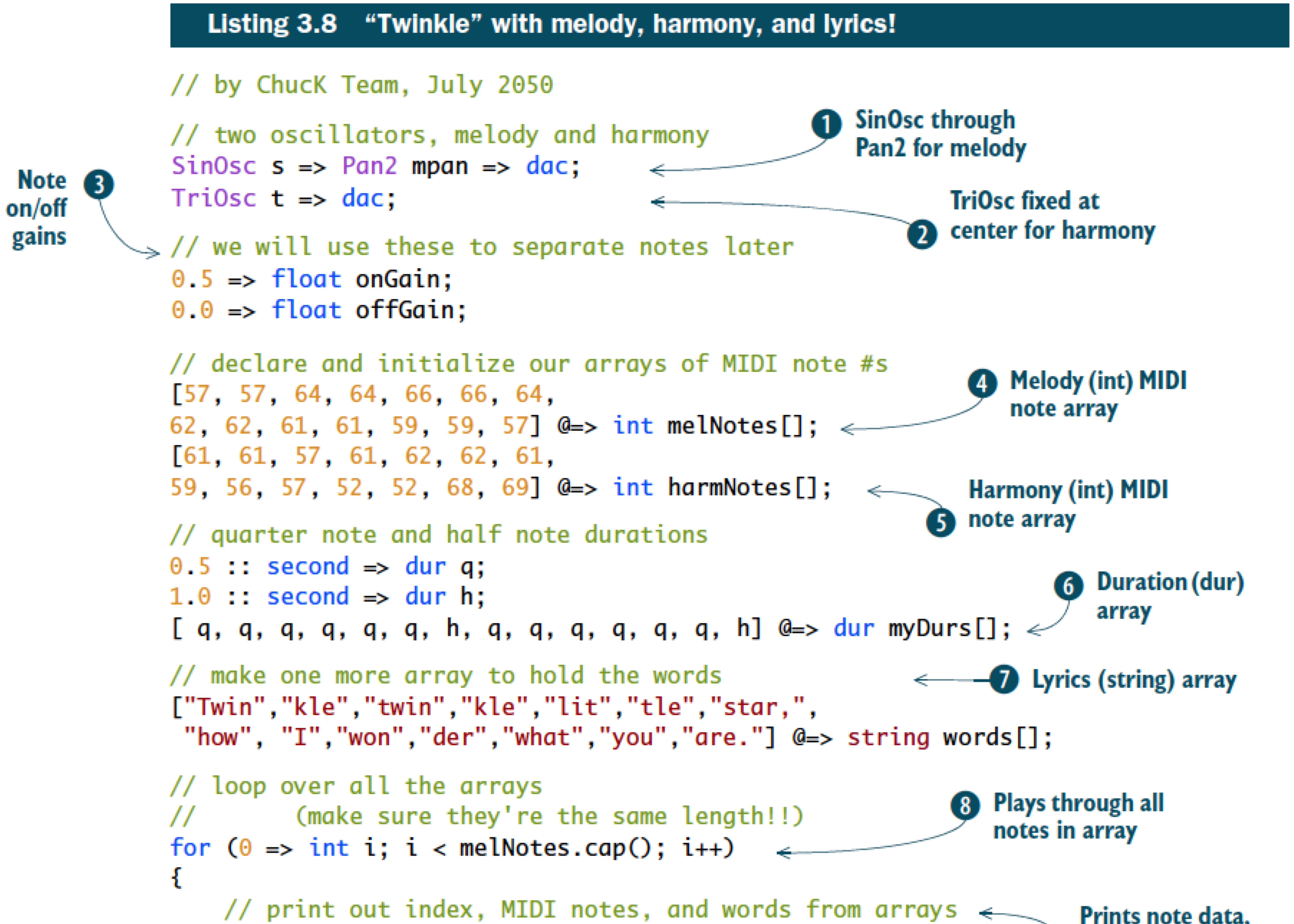


Listing 3.7 An array of strings (the “Twinkle” lyrics)

```
// make an array to hold words and syllables
["Twin","kle","twin","kle","lit","tle","star,",
 "how", "I","won","der","what","you","are."] @=> string words[];
```

1. Declare and initialize array of strings for lyrics.
2. Object copy form of ChucK operator.
3. ChucK figures out how big to create words[] array.

Example: a song with melody, harmony, and lyrics!



Listing 3.8 "Twinkle" with melody, harmony, and lyrics!

```
// by ChucK Team, July 2050

// two oscillators, melody and harmony
SinOsc s => Pan2 mpan => dac;
TriOsc t => dac;

// we will use these to separate notes later
0.5 => float onGain;
0.0 => float offGain;

// declare and initialize our arrays of MIDI note #s
[57, 57, 64, 64, 66, 66, 64,
62, 62, 61, 61, 59, 59, 57] @=> int melNotes[];
[61, 61, 57, 61, 62, 62, 61,
59, 56, 57, 52, 52, 68, 69] @=> int harmNotes[];

// quarter note and half note durations
0.5 :: second => dur q;
1.0 :: second => dur h;
[ q, q, q, q, q, q, h, q, q, q, q, q, q, h] @=> dur myDurs[];

// make one more array to hold the words
["Twin","kle","twin","kle","lit","tle","star,",
 "how", "I","won","der","what","you","are."] @=> string words[];

// loop over all the arrays
//       (make sure they're the same length!!)
for (0 => int i; i < melNotes.cap(); i++)
{
    // print out index, MIDI notes, and words from arrays
```

1 SinOsc through Pan2 for melody

2 TriOsc fixed at center for harmony

3 Note on/off gains

4 Melody (int) MIDI note array

5 Harmony (int) MIDI note array

6 Duration (dur) array

7 Lyrics (string) array

8 Plays through all notes in array

Prints note data,

```
// declare and initialize our arrays of MIDI note #s
[57, 57, 64, 64, 66, 66, 64,
62, 62, 61, 61, 59, 59, 57] @=> int melNotes[];
[61, 61, 57, 61, 62, 62, 61,
59, 56, 57, 52, 52, 68, 69] @=> int harmNotes[];

// quarter note and half note durations
0.5 :: second => dur q;
1.0 :: second => dur h;
[ q, q, q, q, q, q, h, q, q, q, q, q, q, h] @=> dur myDurs[];

// make one more array to hold the words
["Twin","kle","twin","kle","lit","tle","star,",
 "how", "I","won","der","what","you","are."] @=> string words[];

// loop over all the arrays
//       (make sure they're the same length!!)
for (0 => int i; i < melNotes.cap(); i++)
{
    // print out index, MIDI notes, and words from arrays
    <<< i, melNotes[i], harmNotes[i], words[i] >>>;

    // set melody and harmony from arrays
    Std.mtof(harmNotes[i]) => s.freq;
    Std.mtof(melNotes[i]) => t.freq;

    // melody has a random pan for each note
    Math.random2f(-1.0,1.0) => mpan.pan;

    // notes are on for 70% of duration from array
    onGain => s.gain => t.gain;
    0.7*myDurs[i] => now;

    // space between notes is 30% of array duration
    offGain => s.gain => t.gain;
    0.3*myDurs[i] => now;
}
```

4 **Melody (int) MIDI note array**

5 **Harmony (int) MIDI note array**

6 **Duration (dur) array**

7 **Lyrics (string) array**

8 **Plays through all notes in array**

9 **Prints note data, including lyrics**

10 **Sets frequencies from array MIDI notes**

11 **Random pan for melody oscillator**

12 **Turns on both oscillators**

13 **70% of array duration is note on time**

14 **30% of array duration is off time**