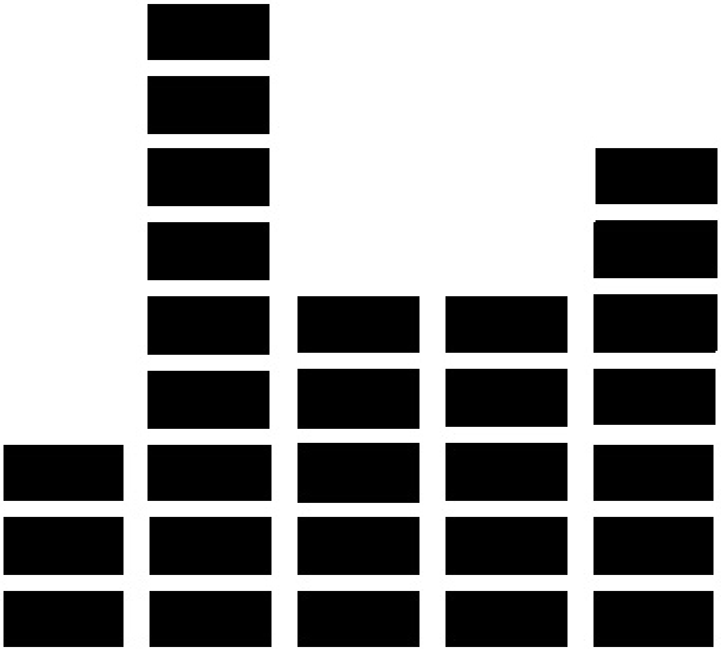
COMS 4115 Programming Languages and Translators
Fall 2013
Professor Edwards

Lullabyte



Stanley Chang (cc3527), Louis Croce (ljc2154),
Nathan Hayes-Roth (nbh2113), Andrew Langdon (arl2178),
Ben Nappier (ben2113), Peter Xu (px2117)

Executive Summary - Lullabyte is a programmatic abstraction of notes in music space. Basic rules and operators, used in conjunction with functions and control flow operators, will allow the developer to create MIDI music files using syntax similar to that of C and C-derived languages.

Intended Uses

-Composition-

Lullabyte allows developers to create musical compositions, either directly or algorithmically. Developers can create music through programs that directly manipulate a song's common elements (pitch, melody, rhythm, etc.). Alternatively, users can develop more complicated compositions by writing programs that generate melodies algorithmically. Lullabyte achieves this by generating simple Java code that will create the MIDI file of the programmed song through an external library.

-Inspiration-

Musicians, like all artists, seek inspiration from their surroundings. Some artists use dice rolls to create melodies and rhythms. However, often times, doing so creates chaotic chord progressions and unpleasant sounds. Furthermore, the notes need to be written down and played in order for the artist to analyze the quality. Though Lullabyte can be used to create specific music, it is also a great tool to inspire new ideas. Since contemporary music is often based on simple chord progressions, it can be coded easily and put into a loop and set as an independent sequence. A randomly generated melody can be created as a separate sequence that chooses notes along a random walk which belong to the chord progression during a specific section. This is analogous to a saxophone improvisation over a bass line in a solo section of a jazz chart. The structure of the partially random melody can be set to reduce chaotic rhythms and general noise in the section. Musical motifs and themes can be reused to create pleasant patterns recognized by humans. With the set structure in place, one can quickly generate multiple MIDI files for inspiration.

Data Types

Type	**Description**
int	An integer value. Example: 3
pitch	Corresponds to a frequency and is represented as one of the 12 musical tones combined with an octave value. Example: `A5`
chord	A chord is a collection of one or more pitches surrounded by square brackets Example: `[C4 G4 C5 ...]`
duration	A measure of how long pitches play with respect to the measure Example: `1/4`
amplitude	Loudness; Integer value between 0 and 100 Example: 42

sound	A combination of chord, duration, amplitude separated by colons. If the amplitude is not specified, the default is 50. Example: `[C3 G4]:1/2:75, [A4 C5 E5]:1/3`
sequence	A list of sounds surrounded by parentheses and separated by commas. Example: `([C3]:1/2, [A4 E5]:1/4, [A4 E5]:1/4, …)`
track	A track is an array of sequences, and dictates the order in which the sequences are rendered to MIDI. Tracks are surrounded by |'s with the sequences separated by commas. Example: `|([C3]:1/2, [A4 E5]:1/4), ([A4 E5]:1/4), …|`
data_type[]	An array of data_types ('data_type' corresponds to one of the above data types)

Symbols

Symbol	**Use**
/	Used with duration.
:	Used to separate chord from duration from amplitude in sound.
;	Used to designate the end of a command.
( )	Used to define a sequence.
| |	Used to define a track.
[ ]	Used to define a chord or an array.
{ }	Used to define a block of code.

Keywords

Keyword	**Use**
constants	`C0 to B9 (ie C1, C#1, D1, D#1, … A9, A#9, B9), true, false`
statements	`if, else, let, do, while, function`
types	`int, pitch, chord, duration, amplitude, sound, sequence, track, void`

Operators

Operator	**Use Cases**	**Descriptions**
+	integer + integer *x* pitch + integer *x* chord + integer *x* chord + pitch *y* amplitude + integer *x*	Same behavior as in C Increment a pitch by *x* semitones; Error if above B9 Increment all pitch inside chord by *x* semitones; Error if above B9 Chord will include *y*; if *y* already exists in chord, ignore Increase the amplitude by *x*; Error if above 100
-	integer - integer *x* pitch - integer *x* chord - integer *x* chord - pitch *y* pitch *x* - pitch *y* amplitude - integer *x*	Same behavior as in C Decrement a pitch by *x* semitones Decrement all pitch inside chord by *x* semitones; Error if below C0 Chord will remove *y*; Error if *y* is not in chord Returns an integer *z* semitones between *x* and *y* Decrease the amplitude by *x*; Error if below 0
=	integer x = integer y etc.	Assignment operator
<<	duration *x* << duration *y* sound *a* << sound *b* sequence *s* << sequence *t* sequence s << sound a	Returns duration *z* of *x* + *y* Returns a sequence of *a* after *b* Returns a sequence of *t* after *s* Returns a sequence with *a* after *s*
*****	integer * integer duration * integer *x* sound * integer *x* sequence *s* * integer *x* amplitude *a* * integer *x*	Same behavior as in C Returns a duration *x* times as long Returns a sound *x* times as long in duration Returns a sequence of *s* concatenated 4 times Returns an amplitude equal to a times x.

<	integer < integer duration *x* < duration *y* pitch *x* < pitch *y*	Same behavior as in C Returns true if duration *x* shorter duration *y*. Returns true if pitch *x* is lower pitch than *y*
>	integer > integer duration *x* > duration *y* pitch *x* > pitch *y*	Same behavior as in C. Returns true if duration *x* longer duration *y*. returns true if pitch *x* is higher pitch than *y.*
==	integer == integer duration *x* == duration *y* pitch *x* == pitch *y*	Same behavior as in C. Returns true if duration *x* is same as duration *y*. Returns true if pitch *x* is same as pitch *y.*
!	!<boolean_expression>	Logical NOT operator.
&&	<boolean_expression> && <boolean_expression>	Logical AND operator.
||	<boolean_expression> || <boolean_expression>	Logical OR operator.

Standard Library Function

`mixdown(track, track, ...)`	Writes the tracks defined by the programmer to a MIDI file in such a way that they will be played simultaneously

Control Flow

Keyword	Example	Description

if else	`if(<boolean_expression>){`
	`      //Statements`
	`} else {`
	`      //Statements`
	`}`
		Same behavior as in C.

while	`while(<boolean_expression>){`
	`      //Statements`
	`}`
		Same behavior as in C.

White Space and Comments

- Space characters, newline characters, and comments are ignored
- // - Comment to the end of the line
- /* text */ - text will be treated as a comment.

Code Examples

```
/*
 * Creates a new sequence with each sound prior being played 4 times
 * consecutively
 */
function sequence quadruple(sequence a){
      sequence b;
      int i;
      let i = 0;
      while(i < a.length()){
            let b = b << a[i] << a[i] << a[i] << a[i];
            let i = i + 1;
      }
      return b;
}
```

```
/*
 * Creates a sequence which we will use to play a blues solo (a random walk
 * on the blues scale)
 */
function sequence randomWalk(int[] scale, pitch root, int n){
      int i, rand, root_step, r;
      sequence out;

      let i = 0;
      let rand = new Random();
      let root_step = 0;

      while(i < n){
            let r = rand.int(5) - 2; //random int between -2 and 2
            let root_step = root_step + r;
            //if root_step is out of scale’s range, place back in range
            if(root_step < 0){
                  let root_step = scale.length() - 1 - root_step;
            }
            if(root_step > scale.length() -1){
                  let root_step = root_step - scale.length() - 1;
            }
            //increment the root pitch by the random-walk of the scale
            let out = out << (root + scale[root_step]);
            let i = i + 1;
      }
      return out;
}
```

```
// main takes three arguments, x and y together define the time signature and
a bpm defines the beats per minute.


function void main (x, y, bpm){
      chord fMajor, cMajor, cMajor7;
      sequence f_c, baseSeq;
      track jude_progression, bass, funky_blues;
      int n;
      int[] scale;


      //"Hey Jude" chord progression
      let fMajor = [F4 A4 C4];
      let cMajor = [C5 E5 G5];
      let cMajor7 = [C5 E5 G5 Bb5];


      // an F chord and C chord with durations of a quarter note
      let f_c = (F:1/4, C:1/4);


      let jude_progression = |quadruple(f_c),
      (fMajor:1/4,cMajor7:1/4,cMajor7:1/4,cMajor7:1/4),
      quadruple(f_c)|;


      let baseSeq = ([F3]:1/1, [C3]:1/1, [F3]:1/4, [C3]:3/4,
                     [F3]:1/1, [C3]:1/1);

      let bass = |baseSeq|;


      //Writes the song to a MIDI file. Both tracks will be combined
      //such that they will play simultaneously in the output MIDI
      //file.
      //plays hey jude chord progession along with a bass line
      do mixdown(jude_progression, bass);


      let scale = [0 3 5 6 7 10]; //blues scale to be improvised on
      pitch C5; //root note of the scale
      let n = 10; //number of notes in the output sequence
      let funky_blues = |randomWalk(scale, C5, n)|;
      do mixdown(funky_blues); //appends funky blues to the MIDI file
}
```