Rhythm
Proposal

John Sizemore	jcs2213
Cristopher Stauffer	cms2231
Lauren Stephanian	lms2221
Yuankai Huo	yh2532

Overview:

The Rhythm language provides a programmatic way to easily compose music from solos to elaborate symphonies through a novel programming language. By modeling a musical score as a programming language, elements such as tempo, timing, notes, chords, and tracks can be used to programmatically create, edit, and play musical compositions.

Purpose:

By providing an intuitive music composition programming language, students and professionals will be able to easily create compositions ranging from simple looping beats to multi-layered scores. While many music composition packages currently rely on visual or audio cues to author music, one of our goals is to explore a purely language based approach to music composition. Rhythm's marriage of music and programming endeavors to pull the curtain back on what some other fancy visual editors try to offer and instead provide a simple programming model. Many modern music composition programs come with steep learning curves that can discourage or alienate new users; Rhythm attempts to emphasize accessibility through simplicity to appeal to programmers without strong musical foundations while still having something to offer for the seasoned musician well versed in music theory. Additionally, Rhythm could be extended beyond the basic building blocks for creating "hand written" music by provided methods for procedurally generated content using numerous music generation algorithms.

Features:

- Primitive definition for notes, chords, measures and tracks
- Primitive implicit attribute definitions for tempo, time signature, and duration
- Primitive composition allowing:
    - Measures to be constructed out of notes and chords
    - Tracks to be composed of measures
    - Songs to be composed of tracks
- Track and measure control techniques such as loops
- Extensibility to support procedurally generated measures and tracks
- Ability to output results that are playable

Syntax:

Primitives:

Type	**Description**

note	A simple note played by an instrument. Can be initialized by a number or by direct description.
	<u>Examples:</u>
	note middleC = C4;
	note middleC = 261.62;

chord	A set of notes associated with one another. Can be initialized by direct description or a sum of directly described notes and/or numbers.
	<u>Examples:</u>
	chord aMinor = Am;
	chord aMinor = A4 | C4 | E4;
	chord aMinor = A4 | 261.62 | E4;

measure(time signature, tempo)	A structure describing a sequence of notes and chords for one measure. Can be used as a standalone structure for building or playing songs, or maybe be combined with the “track” primitive (see below)
	<u>Example:</u>
	measure(4.4, 80) measureA {
	 note middleC = C4;
	 this.B3 = middleC.q;
	}

track(measures, time signature, tempo)	Sequence of notes and chords. Must be specified in terms of length, tempo, time signature, and the notes/chords within. Similar to a structure in C. All musical space assumed to be empty until the notes/chords specify otherwise; can be thought of as a blank “canvas” ready to be filled with the “paint”.
	<u>Examples:</u>
	track(10, 4.4, 80) trackA {
	 chord aMinor = A4 | C4 | E4;
	 this.M3 = measureA;
	 this.M2.B1 = aMinor.w;
	}

Basic Syntax:

Operator	**Description**
+	Increases a half step from a note/chord.
-	Subtracts a half step from a note/chord.
++	Shorthand for increasing by one half step.
--	Shorthand for decreasing by one half step.
*	Multiplies a frequency in a note; multiplies frequency in all notes within a chord.
/	Divides a frequency in a note; divides frequency in all notes within a chord.
=	Variable assignment.
|	Concatenate notes to chords. Concatenate measures/tracks to be played simultaneously.
!	Remove notes from chords. Remove measures/tracks from a measure/track.
{ }	Indicates the beginning/end of a function, measure, or track
.	Used to reference parts of a measure or track. Measures can reference beats; tracks can reference beats and measures. Also references the duration of notes/chords.
( )	Used to denote priority within compound statements.
;	End of a statement.

Keywords:

Keyword	**Description**
note	“note” primitive
chord	“chord” primitive
measure	“measure” primitive
track	“track” primitive
this	used inside of a measure or track definition; refers to the current track/measure begin modified. Used to denote positions of notes/chords within a measure or track definition.

Standard Library Functions:

Function	**Description**
loop(begin, end, track/measure)	Loops a measure or a track.
song()	The entry point of all Rhythm programs. Similar to “main” in C.
upOctave(number, note/chord/measure/track)	Increases the octave of all notes within the argument by the number of specified octaves.
downOctave(number, note/chord/measure/track)	Decreases the octave of all notes within the argument by the number of specified octaves.
toTimeSig(measure/track,newTimeSig)	Modify time signature of a measure/track.
toTempo(measure/track,newTempo)	Modify tempo of a measure/track.

Example Code:

```
/* note definition*/
note middleC = C4;                      // assignment
note eFlat = middleC + 3;               // increase C by three half steps to become E flat
note F = eFlat++;                       // increase eFlat by a half step to become an F
note c_five = middleC*2;                // increase middle C by an octave

/* chord definition*/
chord aMinor = A4 | C4 | E4;            // concatenation
chord aMajor = (aMinor ! C4) | Cs4      // deletion

/* struct definition*/
measure(4.4,80) measureA = {
        this.B3 = eFlat.q;              // Measure and note referencing.
}                                       // The third beat of the measure is an E flat quarter note.

/* track definition*/
track(10, 4.4, 80) trackA = {
        this.M1 = measureA;
        this.M2.B1 = middleC.w;
        measure(4.4, 80) measureB = {
                this.B2 = F.h;
        }
        this.M3 = measureA | measureB;
}

/* example of song function, similar to “main”in C */
song()
{
        /* define Chord, Note, Measure … */
        note    x;
        note    y;
        chord   cho_1 ;
        chord   cho_2 ;
        measure(4.4, 120)       measureA;
        measure(4.4, 110)       measureB;
        track(4.4, 120)   track_A;
        track(4.4, 120)   track_B;
        track(4.4, 120)   track_C;
        track(4.4, 120)   track_D;
        track(4.4, 120)   track_C;
        track(4.4, 120)   track_C;

        /* note  initialization */
        x = A3;
        y = C1;
        z = x + 2;                              /* add two half steps from note x */
        k = y - 1;                               /* reduce one half step from note y */
        /* chord initialization */
        cho_1 = A4 | C4 | E4 | A3;                  /* Concatenate notes to chord*/
        cho_2 = cho_1 ! A3 ;                         /* Remove A3 from chord*/

        /* measure initialization */
        measureA = {
                this.B1 = cho_1.q;
                this.B3 = cho_2.q;
        }

        measureB = {
                this.B1 = cho_2.w;
        }

        /* track initialization */
        track_A = {
                this.M1 = measureA | measureB;
        }

        // function play: play two music measures
        play(track_A);

        // function loop: repeat 100 times of the track_A
        track_B = loop(1, 100, track_A);
        play(track_B);

        // function upOctave: Increases the octave
        track_C = upOctave(1, track_A);
        play(track_C);

        // function upOctave: Decreases the octave
        track_D = downOctave(1, track_A);
        play(track_D);

        // Modify time signature
        track_E  = toTimeSig(track_A, 8.8);
        play(track_E);

        // Modify tempo
        track_F = toTempo(track_A, 200);
        play(track_F);
}
```