'corgi' Proposal


```
PM:                 Alisha Sindhwani (as4312)
Guru:               Justin Zhao (jxz2101)
Testing:            Melissa O'Sullivan (mko2110)
Infrastructure:     Philippe-Guillaume Losembe (pvl2109)
```


September 24, 2014

Description

corgi is a language centered on music translation, generation, and analysis. It will be able to read in a MusicXML file which are both standardized digital file formats for interpreting music and translate the files into the appropriate data structures. Similarly, a user will be able to generate music directly through the implementation of our musical data structures. These data structures will allow our language to quantitatively analyze and find patterns in music that would be difficult to do manually.

Proposed Uses

corgi's main selling point is its ability to search through music. Our data structures make it easy to identify and return the location of specific instances in a given composition. An example program could be finding the longest subsequence of rising notes. This search functionality will allow users to compare multiple pieces of music. corgi can also be used to programmatically generate music.

Syntax

Similar to Java with the list comprehension functionality of Python. Every statement line ends in a semicolon.

Comments

Single line comments are denoted by #:

```
int x = 2; # the int x is set to the value 2
```

Multiple line comments are included between two sets of 3 apostrophes.

```
‘’’ The comment can span multiple
    lines if denoted with opening and
    and closing delimiters as shown ‘’’
```

Variable Declarations

Variables are declared as in Java. The type declaration precedes the variable name which is then followed by an ‘=’ as the assignment operator, the value to be assigned to the variable, and a semicolon.

```
pitch p = 54;
```

Control Flow

Control flow options similar to Java with brackets required. There is support for if, else, for, and while.

```
int count = 0;
for(int i=0; i < 10; i++) {
    if (count < 5) {
        while (count < 3) {
            count ++;
        }
    }
    else {
        count = 2;
    }
}
```

Name	What	Declaration
string	standard char sequence	
int	standard 32-bit integer	int x = 4;
frac	two integers that represent a fraction	fraction f = <3/4>;
duration	wrapper around fraction	duration d = <7/8>;
pitch	wrapper around integer, this can also be instantiated as 'C+4'	pitch p = 52;
rhythm	a collection of durations	rhythm r = [d, d, d];
chord	a collection of pitch duration tuples	chord c = [(p1, d1), (p2, d1)];
track	a sequential list of chords	track t = [c1, c2, c3];
composition	a collection of tracks	composition c = [t1, t2, t3];

Utilities

corgi will come with three utility functions.

Function Name	What it Does	Usage
print()	print string message to stdout	print("Hello, World")
import()	imports a musicxml file into a composition data type	track = import(filename)
export()	exports a composition into a musicxml file	export(track, filename)

Example Programs

Generate Random Track

```
int trackLength = 0;
track t;

while ( trackLength < 100?) {
    chord randomChord;
    duration randomDuration = randomFraction;
    for(i=0; i < randomlessthan5; i++) {
        randomChord += randomPitchInt;
    }
    t.add(randomChord);
    trackLength += duration from earlier;
}
```

Everything Goes Up an Octave

```
composition c = import("filepath/test.xml");

for (t in fun) {
    for (ch in t) {
        pitches += Music.OCTAVE;
    }
}

export(c, “xml”, “filepath/test2.xml”);
```