muS

Final Report

Project site: http://code.google.com/p/mus/source/browse/
Javadoc: http://mus.googlecode.com/svn/trunk/javadoc/index.html

Team 10:
Farbound Tai
Jonathan Dunn
Irene Alvarado
Taylor Owens
Richard Boyle

May 8, 2011

Table of Contents

1. Introduction

1.1 Motivation

Our language “muS” is a language that is used for music composition and analysis. Traditionally, a musician would compose a song by writing notes at his desired position on a music sheet or use computer software with a Graphical User Interface that simulates this action. For instance, to compose the famous “Twinkle, Twinkle, little star, how I wonder what you are”, a composer would manually write down the notes sequence ‘CC GG AA G FF EE DD C’ and designating each note with it’s corresponding duration and octave on a music sheet. To modify the song, the composer would have to go back to each note and change it’s attributes. If a song has common sub-sequences within the song, these sub-sequences are often manually repeated. In short, any manipulation on a sheet of music must be done manually and individually for each note.

If the composer wishes to increase the octave of a group of notes, he or she would have to locate each of these notes and manually increase their octaves. The same applies to duration and pitch modifications for a group of notes. To construct a song that repeatedly uses common sub-sequences of notes, a composer would have to manually recreate each of these notes. This task can be time consuming when the amount of manipulations to a sequence of notes is large. Our language “muS” was created to solve this problem. Through storing sequences of notes in suitable data structures and providing useful sequence manipulation operators and functions, music can be composed more efficiently and the composer will be able to easily experiment with more ideas and rhythms. By combining mathematical computations with music, “muS” will open a new door every music composer and give them the change to experiment with composition styles that are otherwise infeasible.

After a song has been composed, the first step every composer takes is to analyze the result by listening to the song. The composers can decide on how to modify the song after he has heard the notes played out one by one in a sequential order. Our language “muS” takes musical analysis one-step further by allowing the composers to not only hear the music, but also see the music. The composer can designate sections of the songs that are mellow with lighter colors, use different shapes or graphical models to represent different melodies, or mark common subsequences in the song with specific structures. By allowing the composer to specify the pitch and duration of notes in a song as well as append emotional or information on the melody of the sub-structures to a song, “muS” further expands the possibilities of music composition and analysis. By marrying mathematics, data structures, and visual effects with music, our language “muS” will allow music composers to explore music in ways they could never have done before.

1.2 Description

The combination of mathematics and data structures with music is more natural than it appears at first glance. A sheet of music has a strict hierarchical structure with notes, chords, and sequences. Each song is consisted of a sequence of chords and each chord is consisted of a sequence notes. A note has attributes such as duration, pitch, octave, and the visual attributes that we have included in “muS” for musical analysis. A note can be easily represented as an object in a class with attributes corresponding to pith, duration, etc. The hierarchical structure of music can also be easily captured. We can define a chord as an array of notes, and a song as an array of chords. By array indexing, a group of notes or chords can be easily defined. In “muS”, we’ve also created operators that will appeal to music composers such as concatenation of two sequences, appending notes to chords, or foreach operators that allow the programmer to iterate through the sub-structure. For more information, please refer to Section 2 and 3 for our Language Tutorial and Language Reference Manual.

In additional to musical manipulations, we have also included visual attributes to each note to allow the programmer to output a graphical structure of their design on a sheet of music. This includes a wide-range of predefined shapes and colors that can be assigned to each note object. The output of “muS” is a sheet of music with special shapes and colors to allow the composer to analyze musical visually as well as a MIDI file for audio analysis. For more information, please refer to Section 2 and 3 for our Language Tutorial and Language Reference Manual.

Since a sheet of music possess a strong Object-Oriented characteristic, we decided to compile “muS” into Java. We’re using Jlex as our lexical analyzer and CUP as our syntax and semantics analyzer. For more information, please refer to Section 2 and 3 for our Language Tutorial and Language Reference Manual.

1.3 Problem Space

Our language “muS” aims at solving two related problems in music composition. “muS” aids the task of creating music by making it easy for the programmer/composer to manipulate attributes of every note in the song. When the composition is finished, “muS” allows the composer to visually analyze the results as well as through audio. The first problem space for “muS” consists of data structures and functions that will allow easy manipulation of groups of notes. Through the operators and data structure we have provided, the programmer has the flexibility to create complex computations on a sequence of notes and chords. The second problem space for “muS” consists of designs of useful visual models to represent the melody, emotion, or arbitrary sub-structures for a song. Through our extensive predefined colors and shapes and operators, the programmer has a wide variety of tools to achieve his desired visual design.

2. Language Tutorial

2.1. Hello World Program

```
Sequence sequence1 = new Sequence() ;
display(sequence1, “Hello World”); //display function that will
                                    output a sequence with nothing
                                    in it along with a header
```

The function display works mainly to output a sequence of notes and chords in order to view graphical information (notes with different shapes and colors) on an html file. In this case, a dummy sequence variable has been declared with no notes inside, just so a header with “Hello World” can be output.

2.2 Examples for Type Declarations

2.2.a Note

Notes are represented by integers, using the table above. A Note can be declared in the following way:

```
Note exampleNote = new Note(E, 3, 8);
```

Or

```
Note exampleNote = new Note(4, 3, 8);
```

These two lines of code are equivalent. The first argument specifies the pitch value. The lowest pitch value is 0, which corresponds to C, and the highest pitch value is 11, which corresponds to B. The second argument is the octave argument, which ranges from 0 to 9. The last argument specifies the duration of the note in eighth notes. In this example, exampleNote is a whole note. Using the table, exampleNote will be represented by the integer 52.

2.2.a.i Adding shapes and colors to Note

In addition to the pitch attribute, notes also have a shape, color, and instrument attribute. To set the shape used to graphically represent exampleNote to a diamond, we use the < operator and the literal “diamond”:

```
exampleNote<Shape(‘diamond’);
exampleNote<Color(‘darkblue’);
exampleNote<Instrument(‘baritone sax’);
```

The second and third lines of code are similar to the first, except they respectively assign the color “darkblue” and the instrument “baritone” to exampleNote. Therefore, exampleNote will graphically be a dark blue diamond that has the sound of a baritone sax.

2.2.b Chord

The type Chord can represent a chord of one or more notes. The declaration of a chord looks like:

```
Chord exampleChord = new Chord(exampleNote);
```

The code above initializes a new chord and adds our existing note to it. The following adds a new note to our chord:

```
Note A2 = new Note(A, 5, 4);
exampleChord = new Chord(exampleNote, A2);
```

A declared but uninitialized Note cannot be added to a chord. The following will produce an error:

```
Note B4;
exampleChord = new Chord(exampleNote, A2, B4);
```

Because `B4` was not initialized before added to the chord, `exampleChord` is set to `null`.

2.2.c Sequence

The last type is Sequence. A sequence can consist of notes, chords, and other sequences. It is declared without arguments.

```
Sequence exampleSequence = new Sequence();
```

The << operator is used to add notes, chords, and sequences to an existing sequence.

```
Note n2 = Note(A#, 5, 4);
exampleChord = new Chord(exampleNote, A2);
exampleSequence << exampleChord + n2;
```

Now `exampleSequence` consists of `exampleChord` – which includes `exampleNote` and Note `A2` – and Note `n2`. muS also gives the user the ability to concatenate two sequences.

```
Sequence s2 = new Sequence();
s2 << new Note(F, 1, 6);
Sequence concat = [exampleSequence s2];
```

2.3 Examples for Operators

2.3.a Assignment Operator

The assignment operator is useful for assigning new objects, tempPitch or assigning new names for objects, newPitch.

```
Note tempNote = new Note(C, 4, 2);
Note newNote = tempNote;
```

2.3.b Sequence Assignment Operator

The << assigns Notes and Chords to a Sequence.

```
Sequence seq1 = new Sequence();
Note note1 = new Note(B, 4, 4);
Note note2 = new Note(A, 3, 4)
Chord chord1 = new Chord(note1, note2);
seq1<<note1;
seq1<<chord1;

//seq1 is now a Sequence of note1 followed by chord1.
```

The << operator also allows for use of '+' to add multiple Notes, Chords, Sequences, at once to a sequence.

```
Sequence seq1 = new Sequence();
Note note1 = new Note(B, 4, 4);
Note note2 = new Note(A, 4, 4);
Chord chord1 = new Chord(note1, note2);
seq1<<note1+note2+chord1;
//seq1 is now a Sequence of note1, note2, then chord1.
```

2.3.c Duration of Note Operator

The '^' operator will change the duration to the specified integer value.

```
Note note1 = new Note(C,3, 4);
//Changes note1's duration to a half note.
note1^2;
//Changes note1's duration to an eighth note
note1^8;
```

2.3.d Color, Shape, Instruments Assignment Operators

The ‘<’ operator assigns various attributes in the muS languages including color, shape and instrument.

```
Note note1 = new Note(C, 4);
//Assigns Color green to note1
Color green = new Color(green);
note1 < (‘green’);
//Assigns Shape triangle to note1
note1 < (‘triangle’);
//Assigns Instrument violin to note1;
note1 < (‘violin’);
```

2.4 Examples for Built-in Functions

2.4.a Display Function

The built-in display() function is integral to visual analysis, as it draws the sequence, according to the programmers specifications, in a separate .ly and .html file. It can also draw a header of text in the .html file. The sequence will be displayed on a musical staff, while the header will be displayed as text on a white background.

Ex:

```
    display(exampleSequence, “Hello World”);
//This will draw a staff that contains the sequence, represented
    graphically, with the words “Hello World” above that staff.
```

2.4.b Sequence Functions

2.4.b.i Concatenation

Sequence concatenation can be acheived using the ‘[sequenceName1,sequenceName2]’ operator. This will return a new sequence based on the two specified sequences.

```
Sequence seq1 = new Sequence();
Note note1 = new Note(C, 3, 4);
Note note2 = new Note(D, 3, 4);
seq1 << note1;
seq1 << note2;

Sequence seq2 = new Sequence();
Note note3 = new Note(A, 3, 4);
Note note4 = new Note(B, 3, 4);
```

```
seq2 << note3;
seq2 << note4;

Sequence seq3 = new Sequence();
//seq3 will now be note1 followed by note2, note3, note4.
seq3 = [seq1,seq2];
```

2.4.b.ii Chord Referencing

A chord within a sequence can be referenced much the same way as an array in Java. It is done using the 'Sequence(i)' operator. This will return the chord in position "i" of the sequence without removing that chord.

Ex:

```
Chord newChord =  exampleSequence(3);
//newChord will now be identical to the chord in position 3
```

2.4.b.iii Note Referencing

A note within a chord within a sequence can be referenced using the 'Sequence<i,j>' operator. This will return the note in position "j" from the chord in position "i" of the sequence without removing that chord or that note.

Ex:

```
Note newNote =  exampleSequence<3, 2>;
//newNote will now be identical to the second note from the
//chord in position 3
```

2.4.b.iv Sub-Sequence Referencing

A sub-sequence from within a greater sequence can be referenced two ways. First, you can return all chords from position i to position j with the operator, 'sequence[i:j]'.

Ex:

```
      Sequence subSeq = exampleSequence[3:5];
      //subSeq is now equal to the sequence starting with chord 3
and        //ending with chord 5 in exampleSequence.
```

Second, you can return a selection of chords at positions i, j, and k with the operator, 'sequence[i, j, k]'.

Ex:

```
Sequence subSeq1 = exampleSequence[3, 4, 5]
//subseq1 is identical to subSeq from the previous example.
```

2.4.c Foreach Function

In muS, the programmer can use the built in foreach "loop" to visit every note within a sequence and apply some change to it. The programmer can think of the code foreach(sequence) as an object representing every note within a sequence, and thus note functions can be called on it.

Ex:

```
	foreach(exampleSequence)<'green';
	foreach(exampleSequence)<'triangle';
	foreach(exampleSequence)<'trumpet';
```

This code will assign every note in exampleSequence the color green, the shape triangle, and the instrument trumpet.

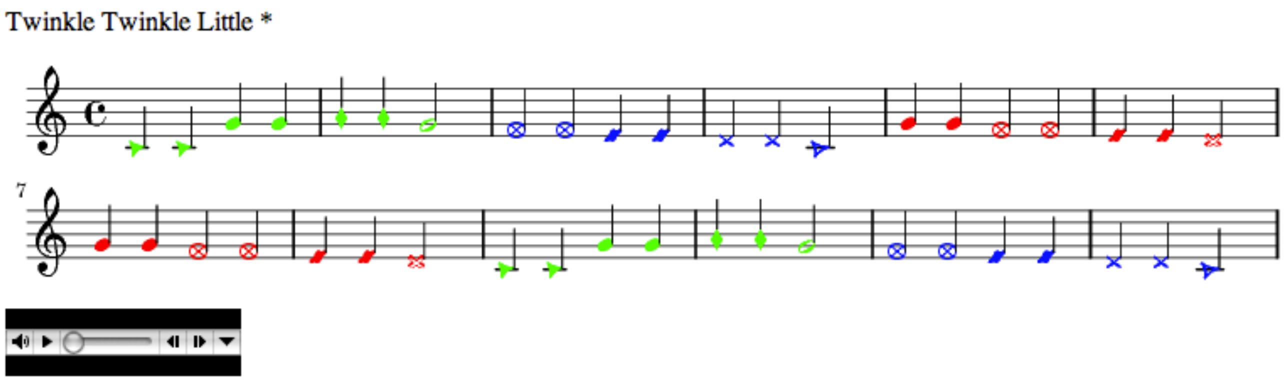
2.5 Final Example

Twinkle Twinkle little star: "Twinkle.mus"

```
//Twinkle Twinkle Little Star

Note C = new Note(C, 4, 4) ;
C<Shape('triangle');

Note D = new Note(D, 4, 4) ;
D<Shape('cross');

Note E = new Note(E, 4, 4) ;
E<Shape('diamond');

Note F = new Note(F, 4, 4) ;
F<Shape('xcircle');

Note G = new Note(G, 4, 4) ;
G<Shape('baroque');

Note A = new Note(A, 4, 4) ;
A<Shape('petrucci');

Note Ghalf = new Note(G, 4, 2);
Ghalf<Shape('baroque');
```

```
Note Chalf = new Note(C, 4, 2);
Chalf<Shape('triangle');

Note Dhalf = new Note(D, 4, 2);
Dhalf<Shape('cross');

Sequence sequence1 = new Sequence();
sequence1 << C + C + G + G + A + A + Ghalf;
foreach(sequence1)<Color('green');

Sequence sequence2 = new Sequence();
sequence2 << F + F + E + E + D + D + Chalf;
foreach(sequence2)<Color('blue');

Sequence sequence3 = new Sequence();
sequence3 << G + G + F + F + E + E + Dhalf;
foreach(sequence3)<Color('red');

Sequence everything = new Sequence();
everything<<sequence1 + sequence2 + sequence3 + sequence3 +
sequence1+ sequence2;

display(everything, "Twinkle Twinkle Little *") ;
```

Generated output:



3. Language Reference Manual

3.1 Project Description

Our language muS is used to graphically represent a piece of music. Our team came up with this idea after reviewing some of the musical languages written last semester. muS is designed for the programmer to compose a segment of music. The output is a MIDI file as well as the sheet music representation of the music that has been composed. Specifically, the sheet music includes the shape and color attributes of each note. With this graphical representation, the programmer can analyze a piece of music in a way that goes beyond merely reading notes on sheet music.

The idea of muS is for the programmer to have the capability of mapping one of the graphical primitives to one of the musical primitives. There are several analytical strategies available, depending on the programmer's mapping. The programmer may assign a specific shape to a given sequence or decide that every note of a certain pitch should have a certain color. Doing so will make that sequence easily recognizable throughout the piece of music. muS is designed to give the user as much freedom as desired while still producing the same output the language is designed to produce: a graphical representation of a piece of music and a MIDI that plays the music.

3.2 Lexical Conventions

3.2.a Tokens

The categories for our tokens are identifiers, operators, constants, keywords, and punctuation symbols. All of the white space characters are ignored. Similar to Java, muS uses braces to mark the beginning and end of a block of code, and semicolons are used to indicate the end of a statement.

3.2.b Comments

muS uses the same style of commenting as Java. /* and */ indicate the beginning and end of a comment. A programmer may right a comment that extends several lines using the /* and */ notations. For one-line comments, // are used. Anything to the right of the // is considered a part of the comment.

3.2.c Identifiers

Identifiers may begin with a letter or an underscore and consist of a sequence of letters, digits, and underscores. Identifiers of the form [A-G] are not allowed because this regular expression is reserved for the pitch literals.

3.2.d Keywords

The keywords for muS are as follows:

Types	**Control**
int	if
Note	else
Chord	for
Sequence	while
void	return
Color	foreach

3.2.e Constants/Literals

muS has several literals that are built into the language. Shapes, instruments, and colors are categories of literals. Shapes – consisting of shapes used to graphically represent sequences – include “square”, “triangle”, “circle”, and “star.” Instruments include the literals that are used to describe the various sounds of different instruments. Examples of these are “violin”, “piano”, “trombone”, and “flute.” The literals for colors are “red”, “green”, and “blue.” Colors also serve as a category of constants as well. The programmer has the ability to declare a new color constant by combining a percentage of each of the literal colors. int literals exist in muS, consisting of integers in a signed 32-bit range.

The pitch literals in muS take the form of [A-G][#][b]. Notes are constants, represented in MIDI by integers using the following table:

Octave	**Note Numbers**											
	C	**C#**	**D**	**D#**	**E**	**F**	**F#**	**G**	**G#**	**A**	**A#**	**B**
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83

6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

3.3 Meaning of Identifiers

Names and identifiers can refer to mostly objects, functions, or type names. An object can sometimes be called a variable and has a location in storage. It depends on two attributes: *storage class*, which is the lifetime in storage of the object, and *type*, which refers to the meaning of the value stored in the object. Finally, the object has a *scope*, which determines where throughout the program the variable is known.

3.3.a Storage Class

Storage classes can be either automatic or static. As in Java and C, automatic variable are local just to a block and are discarded when the program exits the block. All declarations to be found within a block create automatic variables by default, unless otherwise stated.

Instead, static variables can be local to a block or known to all blocks, but they retain their values when the program exits a block or enters a new function. As in Java and C, if a static object is declared within a block it has to be declared with the *static* keyword. If declared outside of all blocks, then it is automatically static, as it will apply to all blocks.

In muS, all variables will be automatically static in the sense that they will be globally scoped and visible to all blocks of the program. They will have static storage duration and are initialized once when declared. This is due to the fact that it facilitates linkage and lets the programmer easily declare variables that can be reused again elsewhere. What this also means is that variable names are unique in muS. No two different variables can be declared with the same name, though a variable can be reassigned to point to a new location.

3.3.b Types

Explained in detail in section 3.4.

3.3.c Scope

Scope defines the area of a program where a variable is visible and begins when the variable is declared. In muS, since all variables will be declared as static, they are visible to all blocks and functions. All variables will be globally scoped.

3.3.d Name Space

A namespace is an environment created to hold a group of unique identifiers. Identifiers are associated with their namespaces, so that an identifier can be independently defined in multiple namespaces. In muS, identifiers must be unique as they will all be relative to one namespace.

3.3.e Linkage of Identifiers

Since all variables will be declared as static and globally scoped, identifiers must be unique in all files of a muS program. Our compiler will generate an error if this is not so.

3.4 Object Types

The language muS supports types of musical notation, numbers, and graphics.

Type	Description
Note	Represents a note. It is comprised of a pitch and a duration (an integer value that represents the length of the note in eighth notes). It also stores variables that represent the instrument the note will be played by, which will be, by default "acoustic grand", “black”, and “default” note shape. Optionally, it can also store variables for color, shape, size, and position, which will be used to represent the note graphically. All these variables will be initialized to null, or possibly some predetermined default, and they can be changed by the programmer. **To declare**: Note exampleNote = new Note(C, 4, 2); //This will be a quarter note that plays the pitch defined by C on octave 4 with duration 2
Chord	Represents any number (1 or greater) of notes played simultaneously, and is considered by our language to be the same as a note when placed into a sequence. It is composed of a list of Notes. **To declare:** Chord exampleChord = new Chord(exampleNote, example1Note, example2Note);
Sequence	Represents any number of Notes, Chords, and Sequences played in succession. It is declared empty and can be built up with the “<<” operator. **To declare:** Sequence exampleSequence = new Sequence();

3.4.a Musical Types

The types Note, Chord, and Sequence make up the musical types used in muS. The basic hierarchy of the musical type is:

Note-->Chord-->Sequence

3.4.b.i Note

The Note type has several attributes. It contains Pitch which is described above. Also, Duration which is an integer value that represents the length of the note in eighth notes. Note also stores variables of instrument, color, size, and shape.

3.4.b.ii Chord

The Chord type is a combination of any number of Note types, that are played simultaneously. This is a list of Notes, where each note has the same duration. Chord also contains a length attribute, which refers to the number of Notes within the Chord.

3.4.c.iii Sequence

Sequence type is comprised of any number of Notes, Chords, and Sequences. The Sequence type also contains the attributes of length. The length of the Sequence refers to the number of Notes, and Chords contained in the Sequence.

3.5. Conversions

Due to the relationship between Note, Chord, and Sequence as detailed above, conversions between these three types are the only ones allowed in muS. Specifically, the order of conversions can go from Note → Chord →Sequence but not the other way around. This is due to the fact that a sequence will represent a collection of Notes, Chords, and other Sequence objects.

3.5.a Conversion of Note and Chord

Note objects can be converted to Chord object and represented as chords with one note. Chord objects cannot be converted to Note objects, even if a Chord object only contains one note. What this really means is that Note objects are used to declare Chord obects and a one note Chord is equivalent to a Note.

```
Note a = new Note(A, 4, 4) ;//Pitch A, Octave 4, Duration 4
Chord c1 = new Chord(a) ;//Chord containing one note
```

The two statements above are basically equivalent. If any of them is added to a Sequence object the same output will occur.

3.5.b Conversion of Note and Sequence

Note objects can be converted to Sequence objects represented as a sequence of only one note. Sequence objects cannot be converted to Note objects, even if a Sequence object only contains one note.

```
Note a = new Note(A, 4, 4) ;//Pitch A, Octave 4, Duration 4
Sequence s1 = new Sequence() ;//a new sequence object
s1 << a ;
```

Now the sequence “s1” contains only one Note.

3.5.c Conversion of Chord and Sequence

Chord objects can be converted to Sequence objects represented as a sequence of only one chord. Sequence objects cannot be converted to Chord objects, even if a Sequence object only contains one chord.

```
Note a = new Note(A, 4, 4) ;//Pitch A, Octave 4, Duration 4
Note b = new Note(B, 4, 4) ;//Pitch B, Octave 4, Duration 4
Chord c1 = new Chord(A, B) ; //A new chord with pitches A B
Sequence s1 = new Sequence() ;//a new sequence object
s1 << c1 ; //The sequence contains the chord c1
```

Now the sequence “s1” contains only one Chord.

3.6. Expressions

The expressions in muS consist of operators in the table below and operands such as Sequences, Chords, and Notes. The expressions are evaluated according to the operand data types with the corresponding operator associativity as shown below.

Operators (in decreasing precedence)		Class	Associativity
Identifiers, constants, parenthesized expression	Primary Expression	Primary	-
()[]<>	Subscripting	Postfix	L-R
<<	Attachment	Binary	R-L
^	Duration Increment	Binary	R-L
[ ]	Concatenation	Arbitrary	L-R

+	Sub-structure Concanenation	Binary	L-R
<	Attribute Assignment	Binary	R-L
=	Assignment	Binary	R-L
,	Comma	Unary	L-R

3.6.a Identifiers, Constants, and Parenthesized Expressions

An identifier is a sequence of alphanumerical letters and underscores. Constants are objects with predefined values. These values cannot be changed. Refer to Section 2 for more information on constants. Parentheses have the highest precedence and parenthesized expressions are evaluated before any other operations.

3.6.b Subscripting, and Member Access

The syntax for subscriptings are denoted by "*var[arg_expression]*", "*var(arg_expression)*", or "*var<arg_expression>*". They represent indexing, subscription of the sub-structure, and the subscription of the sub-structure of sub-structure. For example:

```
seq[i:j]
seq(i)
seq<i,j>
```

Returns the subsequence indexed from i to j, the Chord at the ith location in seq, and the jth Note in the ith Chord in Sequence seq.

3.6.c Attachment

Attachment appends a variable in a lower type hierarchy to another variable. The variable is added at the end of the Sequence. Assignment can also add sequences to other sequences

For example:

```
seq << chord + note
//adds "chord" and "note" behind the last Chord or Note object in
Sequence seq
```

```
Seq1 << seq2 + note + chord
//adds a 'seq2' to another sequences as well.
```

3.6.d Change duration

To change the duration of a note object append an integer: 2,4,8,16,32, or 64 after a '^' symbol. The current value of the note will be substituted by this number.

For example:

```
Note note = new Note(A,4,4) //Note has duration 4, a quarter-note
Note^2 ; //duration has been changed to 2, a half-note
```

3.6.e Attribute Assignment

The attribute assignment operator < is used to assign values to a certain attribute of a variable. For example:

```
n1<Color('green');
//assigns the predefined color green to Note n1

n1<Shape('diamond');
//assigns the predefined shape triangle to Note n1

n1<Instrument('guitar');//ssigns the predefined instrument guitar
to Note n1
```

3.6.e.i Instrument Table

Note note < Instrument('x')		
acoustic grand	tubular bells	clarinet
contrabass	trombone	steel drums
lead 7 (fifths)	fx 5 (brightness)	overdriven guitar
bright acoustic	dulcimer	piccolo
tremolo strings	tuba	woodblock
lead 8 (bass+lead)	fx 6 (goblins)	distorted guitar
electric grand	drawbar organ	flute
pizzicato strings	muted trumpet	taiko drum
pad 1 (new age)	fx 7 (echoes)	guitar harmonics
honky-tonk	percussive organ	recorder
orchestral harp	french horn	melodic tom

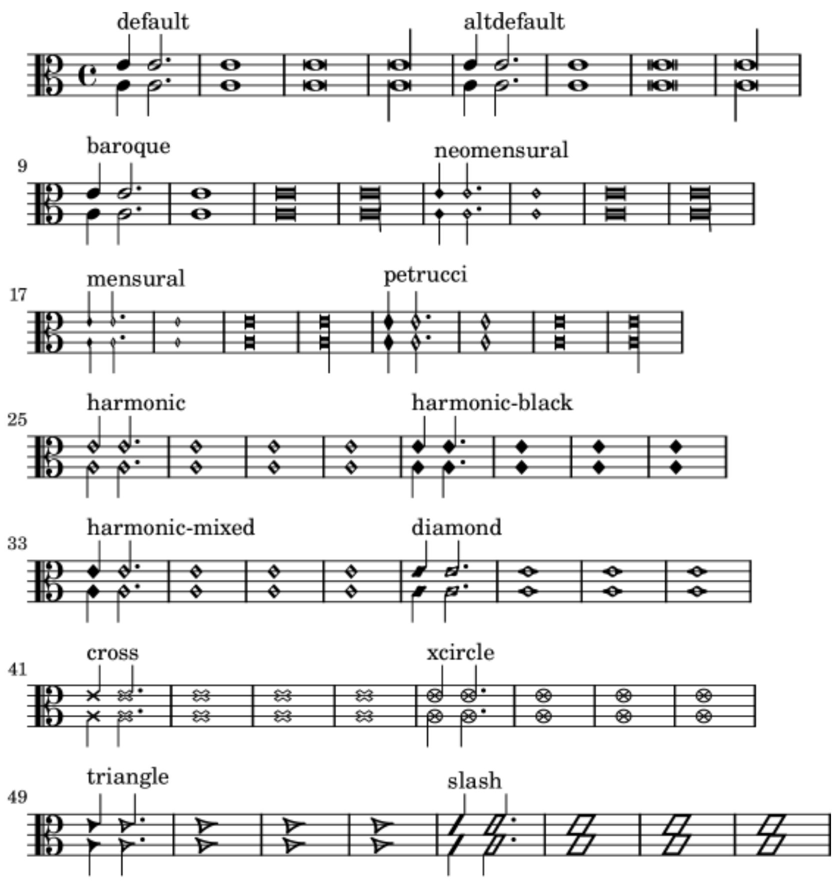
pad 2 (warm)	fx 8 (sci-fi)	acoustic bass
electric piano 1	rock organ	pan flute
timpani	brass section	synth drum
pad 3 (polysynth)	sitar	electric bass (finger)
electric piano 2	church organ	blown bottle
string ensemble 1	synthbrass 1	reverse cymbal
pad 4 (choir)	banjo	electric bass (pick)
harpsichord	reed organ	shakuhachi
string ensemble 2	synthbrass 2	guitar fret noise
pad 5 (bowed)	shamisen	fretless bass
clav	accordion	whistle
synthstrings 1	soprano sax	breath noise
pad 6 (metallic)	koto	slap bass 1
celesta	harmonica	ocarina
synthstrings 2	alto sax	seashore
pad 7 (halo)	kalimba	slap bass 2
glockenspiel	concertina	lead 1 (square)
choir aahs	tenor sax	bird tweet
pad 8 (sweep)	bagpipe	synth bass 1
music box	acoustic guitar (nylon)	lead 2 (sawtooth)
voice oohs	baritone sax	telephone ring
fx 1 (rain)	fiddle	synth bass 2
vibraphone	acoustic guitar (steel)	lead 3 (calliope)
synth voice	oboe	helicopter
fx 2 (soundtrack)	shanai	violin
marimba	electric guitar (jazz)	lead 4 (chiff)
orchestra hit	english horn	applause
fx 3 (crystal)	tinkle bell	viola
xylophone	electric guitar (clean)	lead 5 (charang)
trumpet	bassoon	gunshot
fx 4 (atmosphere)	agogo	cello
	electric guitar (muted)	lead 6 (voice)

3.6.e.ii Color Table

Note note < Color('x')
black
darkyellow
green
red
white
yellow
darkred
darkgreen
grey
cyan
blue
darkblue
darkmagenta
darkcyan
magenta

3.6.e.iii Shape Table

Note note < Shape('x')
default
altdefault
baroque
neomensural
mensural
petrucci
harmonic
harmonic-black
harmonic-mixed
diamond
cross
xcircle
triangle
slash



Credit: picture from LilyPond site

3.6.f Comma

Commas are used as separators for elements in muS.

3.6.g Summary Table

Operator	Description

var = expr
 	
return: type of 'expr'
	Simple assignment operator

Sequence << Chord Sequence << Note Sequence << Sequence Sequence << Chord1 + Note + Chord3 + Sequence2	Adds a Chord to an existing Sequence **Ex:** sequence1 << chord1 sequence1 << chord1 + note2 + chord3 + sequence2

return: Sequence	

Note^int
return: nothing
	Changes the duration of the note to int
	Ex: note1^2 ; //note2 is now a half-note
	note2^8 ; //note2 is now an eight-note

Note < Color('color') **return: nothing**	Assigns a color to a Note. **Ex:** note1 < Color('green') ;
Note < Shape('shape') **return: nothing**	Assigns a Shape to a Note. **Ex:** note1 < Shape('triangle')
Note < Instrument('instrument') **return: nothing**	Assigns an Instrument to a Note. **Ex**: note1 < Instrument('guitar')

3.7 Declarations

3.7.a Type Specifiers

Type specifiers listed below:

```
object_types ::=
      note
      | chord
      | sequence ;
```

3.7.b Custom Types

Custom types are not allowed in muS. The built-in types provide all functionality necessary to create and analyze a piece of music, both musically and graphically.

3.7.c Type Qualifiers

Types cannot be declared mutable or immutable by the programmer. Each type is immutable except for note, chord, and sequence.

3.7.d Function Declarators

muS does not allow custom functions to be declared by the programmer. All functions necessary to build and analyze music graphically are built into the language in the form of built-in functions.

3.8 Statements

Statements are executed in sequence and are executed for their effect. They do not have a value. They can take the following form:

```
expr_list ::= expr_part expr_list //(1)
            | expr_part ; //(2)
```

"expr_list" represents what a valid statement can be. It can either be one expression represented on line (2) or a list of expressions represented on line (1).

3.8.a Empty Statements

muS does not allow empty statements. A minimum file has to have a valid command or token.

3.8.b Expression Statements

Most statements are expression statements and they have the form:

```
expr_part ::=
            object_types    //declaration:   Note,   Chord,
                            Sequence
            | assign        //Assign Color, Shape,
                            Instrument to Note
            | var           //Assignment to a variable
            | functions ;   //built-in functions
```

These statements can be assignments of variables, function calls, control, declaration of object types, or specifically graphical assignments of color, shape, instrument to Note objects; they are called because of their side effect. These side effects are always completed before the next statement is called.

3.8.c Oject Type Statements

```
object_types ::= note | chord | sequence ;
```

Object types can be declared to be notes, chords, or sequences. Each one follows the specific declaration described in the Type section. A detail of the grammar can also be viewed in the Appendix.

3.8.e Assign Statements

```
assign
```

The assign statements refer to expressions of the form "Note < Attribute('attributeName') ;" where Attribute can be: Color, Shape, or Instrument. These are the expressions that allow graphical information to be associated to notes. A detail of the grammar for these expressions can be found in the Appendix.

3.8.e Variable Statements (Initialization)

```
var
```

The variable statements refer to many of the different ways a variable can be declared or assigned to another variable. There are four basic cases:

Variable has not been declared before:
1. A variable is declared as an object type and initialized.

```
Note exampleNote = new Note(C, 4, 4);
```

2. A variable is declared but not initialized.

```
Note exampleNote ;
```

Variable has been declared before:
3. A variable that has already been declared is initialized to something new.

```
exampleNote = new Note(B, 6, 2);
```

4. A variable that has already been declared is assigned to something new.

```
exampleNote = B ; //B and example not have both already been
                       declared
```

3.8.f Built-in Functions

Function	Description

Sequence(int)
 	
return: Chord
	Returns Chord at index int
	
	Ex: sequence1(2) ;

Sequence<int1, int2> **return: Note**	Returns int2th Note from Chord at int1 **Ex:** sequence1<1, 2> ; //returns 2nd note from at position 1
Sequence[int1:int2] **return: Sequence**	Returns new Sequence consisting of all chords from position int1 to position int2. (int2 can be "end" to allow access from position int1 to end of the Sequence.) **Ex:** sequence1[2: 6] ; // returns sequence of Chords2-Chords6
Sequence[int1, int2, int3...] **return: Sequence**	Returns new Sequence consisting of all chords in position int1, position int2 and position int3. **Ex:** sequence1[1,3,5] ; //returns sequence of Chord1+Chord3+Chord5 ;
[Sequence1, Sequence2] **return: Sequence**	Concatenates two Sequences **Ex:** sequence1 = [sequence2, sequence3] ;

foreach(Sequences)<Color('color');
 	
foreach(Sequence s)<Shape('shape');
 	
foreach(Sequences)<Instrument('instrument');
	This control is the most important in our language. It is implemented as a function. It iterates through all elements of a Sequence. It iterates through every single note in a sequence, including those inside chords and applies an attribute to them.
	
	ex: foreach (exampleSequence)<Color('green');

display(Sequence sequence1, "header")	Writes lilypond code to .ly and .html files to display them. Takes in a header string.

4. Project Plan

4.1 Project Roles

Project Manager: Irene Alvarado
System Architect: Taylor Owens
Verification and Validation: Farbound Tai
System Integrator: Richard Boyle
Language Guru: Jonathan Dunn

4.2 Project Timeline

Date	Assignment
February 23	White paper due
February 24 - March 7	Define lexical conventions, types, operators
March 8 - March 21	Define expressions, comparisons, assignments, initialization, selection statements, etc.
March 23	Language tutorial and reference manual due
April 4	Finish lexical analyzer and finish most of the back end graphical output of notes, colors, shapes
April 11	Finish main structure of the parser and test lexical analyzer extensively
April 25	Finish most of the grammar of the parser
May 2	Extensively test the parser as well as the integration of components
May 6	Finish final report
May 10	Project presentation

4.3 Communications

Apart from the group's development environment, which we started to use extensively once we started coding, there was a question of how to stay in touch and exchange quick ideas when writing the white paper, reference manual, and language tutorial. To do this, we created a google groups that can be accessed here: https://groups.google.com/forum/#!forum/programming-languages-and-translators

We used this space to discuss details of our language when we could not meet in person. In these cases, we used Google Docs extensively to edit the same documents. Otherwise, the team held weekly meetings every Monday from 7-9 in which we would discuss our plan of action for each week. The team leader would send weekly emails with to do lists and with the review of what was discussed for the week.

4.3.a Project Title Thread

me(Irene Alvarado change)   Feb 22
use this space to suggest titles.

First choice for now: MusiCode?

me(Irene Alvarado change)   Feb 22
MusiGraph? (Kind of lame but it has the graphical part...)

me(Irene Alvarado change)   Feb 22
MusiGraph? (kind of lame but has the graphical part?)

R Michael Boyle   Feb 22
shapeSong? lol

Farbound   Feb 22
I thought we could also call it Musical, since it's combines
visualization with music, just like a musical theatre...

Jonathan Dunn   Feb 22
my vote is for MusiCode, although ShapeSong is fun to say; it sounds
like a tongue twister. hehe. first time i read it, i pronounced it
"ShapeShong." lol.

me(Irene Alvarado change)   Feb 22
In any case, I would say MusicL so it sounds like "musical" but isn't just written like it. I still like MusiCode too, I think.

R Michael Boyle   Feb 22
muS
cheesy but like its your muse, S for shape and got the mu for music ....

Taylor Owens   Feb 23
YES! muS all the way. That's where my vote is. Although I like MusicGraph, too

Jonathan Dunn   Feb 23
muS works for me!

Farbound Feb 23
I like muS too =D

R Michael Boyle Show activity Feb 22
shapeSong? lol

4.3.b White Paper Thread

me(Irene Alvarado change) Feb 19
Hey, we can use this space to discuss changes, comments, additions to the white paper.

R Michael Boyle Feb 20
Everyone should have access to my portion, feel free to edit or give me feedback.

me(Irene Alvarado change) Feb 20
Its kind of funny, the only name the spell check wants to change is Farbound. :)

Jonathan Dunn Feb 21
Hey guys. I posted my portion of the white paper. I mentioned a couple of the things we discussed in our last meeting. Feel free to let me know if I left out something or should remove something. As far as the operators are concerned, I just mentioned two operators that I thought would be a good idea to have. But I can definitely add any operators we decided to use. Also, I used "L" as the substitute for our language's name. So whenever we come up with a name, I'll replace all the L's.

Jonathan

Farbound Feb 21
Haha I know. I have that problem whenever I get a new computer too.

me(Irene Alvarado change) Feb 22
Hi everyone, I've added a final section to my part (challenges and additional ideas) sorry i took so long to post :\

me(Irene Alvarado change) Feb 23
FINAL version of the whitepaper (even if called V1). I've posted it here in case you guys want the real doc file.

repeated: You guys can still take a final look and post comments (on the online version

which should be available to all of you) if you
want. I will upload it tomorrow by 11 or 12 so there's time for final
edits. I don't think we need much though, this white paper kicks ass!!

me(Irene Alvarado change) Show activity Feb 23
Final update: our white paper is posted. you can download the pdf version from the class files on courseworks. Thanks for the good work!

4.3.c Language Tutorial Thread

me(Irene Alvarado change) Mar 22

Jonathan Dunn Show activity Mar 23
Question:

if the programmer is able to declare a note like:

```
Note n1 = new Note(C, 2);
```

or

```
Note n2 = new Note(C#, 3);
```

then why do we need a Pitch type? can't we just have all of the pitch
values declared as built in literals?

Jonathan

me(Irene Alvarado change) Mar 23
hadnt we said pitch was not a type and would be built in like you said?
help others, refresh what we talked about?

R Michael Boyle Mar 23
I think we said that you could do both, Note n1 = new Note(C, 2) uses
the built in pitch, where C is the note and 2 is the number of eighth
notes. But if we declare a pitch we can define and octave and a note.

Jonathan Dunn Mar 23
reading our definition of a pitch from the white paper, the programmer declares a pitch using a value from 0 to 11, which corresponds to the scale C through B. if the programmer defines his own pitch, what other values would he use? I am wrong to assume the only values a pitch can have is C through B?

me(Irene Alvarado change) Mar 23
no, you are right. i think we can leave it as pitch in letter is equal to pitch in numbers as it can be built in. essentially they correspond to the same thing.

in Both cases we need to specify the octave And the duration.

```
note n = new note (C, octave o, duration d)
```

or

```
note n = new note (1, octave o, duration d)
```

is this correct?

R Michael Boyle   Mar 23
That looks fine to me.
But can we leave out octave and/or duration if we want it set to some default?

me(Irene Alvarado change)   Mar 23
I think we can probably do that but maybe with an additional function for the user to specify. no? for that maybe it's better not to include I. the reference and then add it at the end?
dunno

Jonathan Dunn   Mar 23
I was thinking of the same thing, but i think we need all three. i think if we kept all three, it would make our jobs easier. if we exclude one of them, we would have to write a way for the user to change it later. personally, i think it would make thinks a bit simpler if we just make the user define all three.

R Michael Boyle   Mar 23
Yeah, it would simplify on our end...let's just do that.

me(Irene Alvarado change)   Mar 23
One final questions that has come up as I am joining parts. Since we took off area and everything, we havent exactly defined ho to play and map a sequence.

Two options I think:

```
sequence1.play();
sequence1.display() OR sequence1.map() ;
```

OR

```
play(sequence1);
display(sequence1) OR display(sequence1) ;
```

You see the difference right? One is a built in function, the other one is a function of an object.

Jonathan Dunn Mar 23
what would the map function do?

R Michael Boyle Mar 23
built in function type...I think

R Michael Boyle Mar 23
Are we going to have a built in print function? Could we use display
to somehow display text for helloworld?

R Michael Boyle Mar 23
Map would draw the whatever shapes/colors defined
On Wed, Mar 23, 2011 at 8:53 PM, Jonathan Dunn <dunn...@gmail.com> wrote:

me(Irene Alvarado change) Mar 23
for hello world just print normally i think right? Like java print? Or I guess we could display like a header on our display. So that could include hello world

like

header(String) something like that.

Or in our display function include a space for the header string

So:

display(String header, Sequence sequence1);

R Michael Boyle Mar 23
I just used display with a header...and null for the sequence, is that okay?

Jonathan Dunn Mar 23
can we just define two display methods: one that takes a String as an argument, and one that takes a Sequence as an argument?

me(Irene Alvarado change) Mar 23
yeah i think so. So it would be a built in standard function i guess:

display(Header, Sequence);
play(Sequence)

Jonathan Dunn   Mar 23
i was just thinking:
display(String);
display(Sequence);

me(Irene Alvarado change) Mar 23
yeah, also yup. cool.

Taylor Owens   Mar 23
Hey guys, just letting you know, I have sequence.display() and sequence.map() in my examples. Don't know if anyone has seen that, yet.

Taylor Owens
CC '11
Computer Science/Mathematics

Taylor Owens Show activity Mar 23
It could easily be changed to that format, I was just trying to keep it consistent with sequence.length()

Taylor Owens
CC '11
Computer Science/Mathematics

me(Irene Alvarado change) Mar 23
dont worry, im correcting for those things.

4.3.d Language Reference Thread

me(Irene Alvarado change) Mar 22

Jonathan Dunn   Mar 22
(I apologize for the duplicate; I posted my question in the wrong section the first time.)

I am confused about including Shape, Color, and Instrument as keywords. I know we decided to eliminate these as types, and instead to just say "square" or "piano." But wouldn't these shapes and instruments have to be defined somewhere? The programmer can't just say "piano" and expect the sound of a piano if we haven't defined the relationship between the string and the sound. And should the specific colors, shapes, and instruments that we're using be keywords as well? Or is that unnecessary?
I hope my question makes sense.

Jonathan

me(Irene Alvarado change)   Mar 22
sure, its a good question. I think we agreed they would be built in constants? Like C, A#, etc. which can be found in the midilc reference manual. They go under the constants/literal section because we will just build in the corresponding java function for these cases.

So 'piano', 'square', 'blue' are all literals. So Instrument and Shpae would not be keywords (they are not types)

Color is still a keyword because I thought we said we could define it in terms of rgb as well. (color is still a type)

Is this correct?

me(Irene Alvarado change)   Mar 22
Hey guys, i've added almost all of my parts for the reference manual. If you want to make comments, take a look. I'm just missing like two mini-sections in the end, to be done later today.

Have fun with your parts!

Jonathan Dunn   Mar 22
Correct. Okay. I guess my question should have been: are Shape and
Instrument still tokens? I'm thinking they'd look like

Shape = ["square" , "circle", "triangle"]
Instrument = ["piano", "drum", "trumpet"]

Of course these are just examples, and we would have to decide all of
the shapes and instruments we're using. I know how to incorporate them
into the reference manual. I guess I'm more confused as to how they
fit into the lexical analyzer.

Jonathan

R Michael Boyle   Mar 22
I uploaded a rough draft of the types, some of them I'm pretty vague
and unsure about how to elaborate on them...sequence being a major
one..what do you guys think?

Jonathan Dunn   Mar 23
I have posted the Lexical Conventions portion of the Reference
Manual. For some reason, the table got screwed up when Google

converted it. Irene, I can email you the copy I have so you can use
the original table when compiling all the pieces.

jD

me(Irene Alvarado change) Mar 23
hi guys, I was going over my parts for the reference manual. I think we should add two built in funcitons:

Sequence.length() //returns number of chords and notes in the sequence
Chord.length() //returns number of notes in the sequence.

Jonathan Dunn Mar 23
sounds good to me.

R Michael Boyle Mar 23
I'll update object types then.

Taylor Owens Mar 23
And I'll add them to my examples
Taylor Owens
CC '11
Computer Science/Mathematics

Taylor Owens Mar 23
It seems like we might want a note.length that returns the length of a note, too. I've added an example in my section of what I think that will look like, but it's easily removed. What do you think?
Taylor Owens
CC '11
Computer Science/Mathematics

me(Irene Alvarado change) Mar 23
Oh like for reference you mean?

Taylor Owens Mar 23
Right. If the programmer wants to change all whole notes to be green, they need a way to do that.
Taylor Owens
CC '11
Computer Science/Mathematics

me(Irene Alvarado change) Mar 23
right.

Jonathan Dunn   Mar 23
i think it's a great idea. i have just one concern: in the declaration of a note, the duration is defined in eighth notes, but the length function returns the duration in sixteenth notes. shouldn't they be the same? i think we should changed the declaration to sixteenth notes to match the length functions.

me(Irene Alvarado change) Mar 23
yes, they definitely should. eight-notes is probably better.

Jonathan Dunn   Mar 23
sounds good to me.

Jonathan Dunn   Mar 23
hey guys,

i'm just about done with my portion of the tutorial. i just have one small question: with the sequence concatenation operator

[Sequence1 Sequence2]

is the resulting sequence stored in sequence1 or do does the user have to use it like such:

Sequence s2 = [Sequence1 Sequence2];

R Michael Boyle   Mar 23
Sequence s2 = [Sequence1 Sequence2];
is how I did my examples.

me(Irene Alvarado change) Mar 23
make comments on the reference manual i posted if there is anything terribly wrong. If not, just waiting to add part 6 and were done.

Jonathan Dunn   Mar 23
is the expression document that farbound posted not complete?

me(Irene Alvarado change) Show activity Show recipientsMar 23
not after 5.7

5. Language Evolution

The team faced some problems throughout the beginning of the semester with regards to the syntax and specific functions that we would allow for the language. Most of the problem relied on the fact that we were not sure how to implement the back end part and how dynamic, versatile and nice our graphics would look. There was also some misunderstanding as to how we would implement the conversion to MIDI. For this reason, we were not so sure how powerful our language could be or how useful the graphical part could turn out to be. It was not until the team started to discover some very useful tools for graphical editing of music and researching how to create MIDI files that we were able to specify in a much clearer way what our language would include and how to represent it.

The first major challenge became a design decision regarding what object type would contain graphical information. Should we allow notes to be related to shapes and colors? Or should it be done at the chord level? Would it be useful for Sequence object to store graphical information instead of Note objects? The question was essential to how the programmer would interact with different object types and we ended up deciding that the most functionality would come out of defining the graphical information at a Note level. In muS, each note can be assigned a color, instrument, or shape attribute.

Finally, another design decision came at the moment when we were dealing with memory location of objects and the assignment of different notes and chords to sequences. The problem arose if muS should clone note and chord objects when they are added to sequences so that further changes in color and shapes of the note and chord object would not affect already declared sequences. The other alternative was to make sequence objects point to already declared notes. The difference can be illustrated with an example:

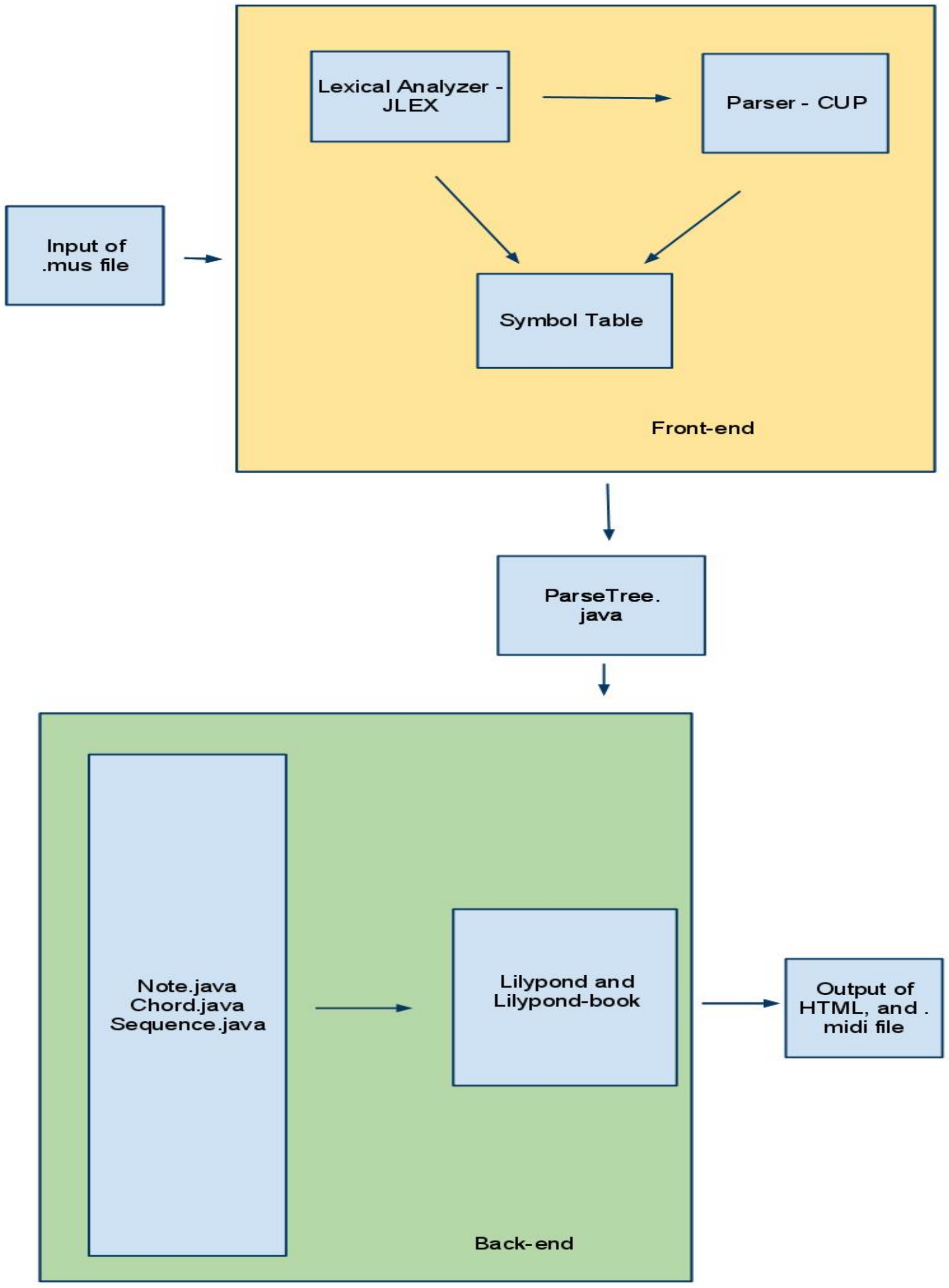
```
Note a = new Note(A, 4, 4) ;
a < Color(‘green’) ;
a < Shape(‘diamond’) ;

Sequence s = new Sequence() ;
s << a ;

a < Color(‘red’) ;

display(s, “example”) ;
```

According to what muS currently implements, we decided to clone note objects when they are added to sequences. In the example above, the sequence would output a note with pitch A in Green. Otherwise, if we had not cloned the objects that are fed into sequences, the sequence would have displayed a note with pitch A and color Red. These two decisions were probably the most important in our language.

6. Compiler Architecture

6.1 Overview & Diagram

The compiler architecture consists of a front-end, back-end, and an output from the back end to lilypond and & lilybook-book. See diagram below:



The source file is fed into the front end, parsed and checked for syntax and then sent to the back end where it creates code to be executed in lilypond.

6.2 Components

6.2.a Front-end

The Front-end part of the compiler contains LEX/CUP. Java-Lex is a lexical analyzer that outputs a java yylex file. Java-lex takes as input the specification of lexical analyzer, including all regular expressions and all symbols. The output of Java-Lex is a java source file that will catch tokens from the input stream of the source file. When an input stream is input to the specification JLex will break itnto tokens and pass it to the parser.

CUP (Constructor of Useful Parser) is a LALR parser for Java. The CUP parser uses an input file with a grammar that was created for the muS language. CUP receives the tokens from the lexical scanner and then uses those tokens to follow the grammar rules.

Java-Lex and CUP both use a symbol table that contains all of the tokens. The The output of JLex and CUP is a ParseTree, created in ParseTree.java which calls the back end.

6.2.b Back-end

The back-end of muS contains objects, of Note, Chord, Sequence, and a reference table. It also contains a file for Error output. The ParseTree will call each object as they are created. A Note object is created for each note in muS, and the same goes for Chord, and Sequence. After each object is created from the ParseTree, the final sequence is output to lilypond. Each note, and chord contains lilypond code, which gets written to a lilypond file for each one.

6.3.c Calling lilypond & lilypond-book

The lilypond file is created from the master sequence and then input into lilypond. Lilypond application is called to read the output of the back end. Lilypond will output just and image file, and lilypond-book will output the html file. It reads the file and outputs a final html file containing the midi music file and an image containing the specified graphics and notes.

7. Development Environment

7.1 Overview

The muS team used various tools to aid in the development of this language. The tools helped in forming the initial ideas, allowed for us to work simultaneously on the same item and control where changes were made. The tools that were used were Google Docs, Google Code, Eclipse, and Lilypond.

7.2 Google Code

Google code was used to keep an svn repository of all code for the muS project. The team used Google code to synchronize the project as it was being created. It also allowed for a site to host a Javadoc of our language.

7.3 Eclipse

All development was done in Eclipse. The back was built using Java and Eclipse was valuable in checking for errors and organizing code. Eclipse also allowed for plugins such as Subversion and CUP/LEX that gave the ability to build the front-end and update our Google Code repository.

7.3.a Subversion

The first plug-in used with Eclipse was Subversion. This allowed for updated files to be committed to the repository and to download changed file. This was also necessary for merging of the files. The plug-in would denote where the changes in each file were and what conflicts they might create.

7.3.b CUP/LEX plug-in

Another plug-in utilized was the CUP/LEX plug-in which would compile the CUP and LEX files within Eclipse into their java compiled forms. This made it easy to update the parser and lexer and simultaneously see the changes with our back end java code.

7.4 Lilypond

Lilypond is a musical typsetting similar to LateX for mathematics. Lilypond allows for sheet music to be created through various textual input. The back end of mus outputs each note, and chord into a Lilypond text file which is fed into Lilypond and Lilypond-book. This then produces a .midi file along with the corresponding sheet music.

8. Test Plan & Test Suites

8.1 Phase I: Lilypond

When checking for errors within our lilypond package, we checked that the graphical representation matched the musical specifications in the code: notes were in the right place on the staff; correct colors, shapes, and instruments were assigned to the correct notes and sequences; and listening for the exact instruments when the music actually played.

8.2 Phase II: Lexical Analyzer

Error checking for the lexical analyzer required paying attention to very minute, particular details. We had to ensure that each lexeme matched the appropriate token. This involved making sure each token was accurately defined, the rules were written correctly, ***and*** the rules were listed in the correct order. Tokens had to be created for letters, numbers, keywords, quoted text, and every symbol that could appear in a muS program. Printing out each lexeme and the token it matched assisted us in error checking along the way.

8.3 Phase III: Semantic Analyzer

The semantic analyzer is the last phase of the back-end of the compiler and is the gateway into the front-end. Therefore, error checking at this stage had to be done thoroughly, and took the longest. The semantic analyzer executed the Java code. In order for the right code to be called at the right time, we paid close attention to how the grammar was written. All of the necessary non-terminals and terminals had to be declared. The terminals corresponded to the tokens provided in the lexical analyzer; we had to ensure for every terminal declared in the semantic analyzer there was a token declared in the lexical analyzer. In order to test the grammar, and test if we covered every possible yet acceptable grammatical syntax case, we wrote and executed simple muS code.

8.4 General Errors

We wrote an Error class that outputs messages for the more common errors: reusing variable names (i.e. declaring a variable that has already been declared), initializing an undeclared variable, operating on an uninitialized variable, using an operator on objects of types it is not defined for, null pointer exception, comparing two objects of different types, and passing an object of one type where another type is expected. We went through every method of our Java code, checking for every possible error within that method, and used the Error class to notify the programmer when an error occurred.

9. Conclusions & Lessons Learned

9.1 Lessons Learned

9.1.a Project Manager – Irene Alvarado

It is tough to work in a team, everyone know this. I thought I did too, because I had done other team projects in the past. This case was very different though. I did not realize that a coding project would be particularly challenging. First, I was having trouble visualizing what the project would really demand, the difficulties that might arise, and the time requirements. Since this was our introduction to compilers, none of us really had a good idea of how to go about starting to implement the language or how to fit the different components together. Second, I found out there was a component to communicating between teammates that went beyond just sending out emails and posting on threads. Since the code would be updated daily, it was hard to keep track and understand what everyone was doing. Small edits could mean that the grammar did not work anymore and for moments when we were not face to face, this was hard to manage. Fortunately, we discovered that using subversion and eclipse could help us out in this respect. By the middle of the semester, I had found a way to make my emails more relevant and helpful, marking things in list format so that they could be easily understood. Finally, I realized that it really is crucial to start early in a project like this, because one can save a lot of time by laying out a general structure before spending time implementing different components. On the other hand, there is only so much one can plan in a project like this, where no one had implemented a compiler before. I wish we had started our implementation earlier to discover any flaws in our plan and have time to correct accordingly.

9.1.b System Architect – Taylor Owens

There are two things every instructor tells you when you begin work on a large project like this one: start early, and plan carefully. We all think we know what those things mean, and we all know our own limits concerning the amount of time we need to dedicate to a project. In a group of five, working on a semester-long project, the importance of following those words of advice increases by an order of magnitude. It's hard enough to follow a strict schedule on your own time, and when you have to coordinate 4 other people's code and input, it becomes nearly impossible. The only way to ensure everything will be ready is plan meticulously.

The most frustrating thing that can happen as a programmer is to write a piece of code that will never see the light of day because of poor planning or a change of design. Unfortunately, because our initial structural design did not take into account some of the final problems that we would run into, that structure became muddled quite quickly, and we were forced to remove code and more robust functions. Maybe it's impossible to foresee everything that will eventually happen over the course of a project's life, but I learned that the initial planning and architecture needs to be done with a firm view of the final implementation in mind. It is not enough to have a high-level structure planned out. If we could do it again, I would devote much more time and energy into the planning phase, making sure to solidify, if not every detail, every structural element before beginning any actual coding.

9.1.c Verification & Validation – Farbound Tai

In my past coding projects, planning and implementation were usually two separate stages. If I sat down and lay out a well-thought plan, then there usually wouldn't be too much trouble involved in implementation. This project was different in many aspects. Since none of us have ever built a compiler, we didn't have a good idea on what to expect. Although we spent enough time discussing the features and syntax for our language, we didn't spend enough time on the grammar for our parser. This was a big problem later in the project. As production rules became more and more complicated, we had to keep changing our code to cover cases we didn't think of in the beginning. Although these problems could have been avoided if we envisioned them earlier and planned better, I believe many of our problems could only be discovered through implementation. Had we begun the implementation as soon as we had some idea on how to build the components of our compiler, we'd have had the time to handle these problems. I've learned that the process of planning and implementation is an iterative process instead of a directed acyclic graph. It is important to both plan well and act early so flaws in the plans can be discovered early in the process.

9.1.d System Integrator – Richard Boyle

A semester long coding project is no easy task I found out this semester. I came into this project thinking that we would have the project completed with a few weeks to spare, that inevitably was not the case.

A major problem that we faced was knowing exactly where to start. We had no problem coming up with the idea and writing the reference manual, but when it came to actually implementing the code we hit a major road block. The back end, where we felt comfortable, writing typical java code was produced easily, but the front end was confusing to say the least. Looking back I wish we would have spent more time planning the grammar. It works as is, but a few functions that we wanted to implement did not because of lack of time, or lack of structure in our parser. We also faced problems coordinating with every one's schedules and not sticking to a set plan, producing things on time.

Things I would do differently would be spend more time in the planning stage and stick to a plan when it is set. Overall though I am happy with our project output, though we did not implement everything we set out to, I believe we created something better than what we initially said we were going to.

9.1.e Language Guru – Jonathan Dunn

Firstly, I will admit that I underestimated how much work the compiler actually does! After diving into a compiler – learning about the various phases and how they are implemented – and writing our own compiler, I have a much greater appreciation for all compilers now. First, there is writing the lexical analyzer, then the semantic analyzer, then the Java code. Not to mention figuring out how to get them to work together. There is so much involved with error checking, and it has to be done everywhere in the code! Deciding along the way what the programmer will and will not be allowed to do was a bit frustrating as well. My appreciation for compilers has grown greatly! I actually learned a lot while working on this project, both technical and nontechnical skills. For starters, effective communication is imperative, especially when five people are working on a limited amount of code – making simultaneous changes – and are dependent on each other to pull their own weight as far as writing a portion of the code. This leads me to my second lesson: meeting frequently and coding together help tremendously!! Even the meetings that only last 15 minutes play their part. It helps to discuss things as a group, gauge where everyone is on the parts they were assigned, and set goals to accomplish by the next meeting. Doing so will make the process easier in the long run. Lastly, I learned how to use a SVN repository. After overcoming the problems I had setting it up I found it to be quite convenient and useful.

I honestly enjoyed working on this project. It was a great yet challenging learning experience and I think we made a great team. I am very proud of the work we have done and the output we have produced. We were a bit overzealous with our ideas of muS in the beginning, but it turned out to be better than I thought it would. As the saying goes: “Shoot for the moon. Even if you miss, you’ll land among the stars.”

9.2 General Conclusions & Challenges Faced

Reading our individual sections, you can see two themes that appear with everyone's notes. First, we all underestimated the actual amount of work that goes into creating a full compiler in a team of five people with different schedules. We all understood, theoretically, that we would need to put in a great number of hours both planning and coding, but it is an entirely different thing to think theoretically about work time and to actually account for hours spent in front of a machine or a piece of paper. Second, we all expressed a need for greater planning throughout the project. We had a very solid idea of what we wanted to accomplish with muS and what our vision for it was, but when it came time to incorporate the actual pieces of the project into the final thing, our plan was much less solidified. As a result, we lost some important time throughout coding when it came to getting each person's work to integrate into the whole which ended with us actually having to cut some functionality that we all wanted to see in our final result.

The best example for this can be seen in our cutting important control-flow functionality from the final project. When it came time to build our grammar in CUP, our approach was to work from the ground up-- first get the lowest pieces working (variables, note & chord declaration) and then move on to incorporate them into the greater design of the grammar. However, because of the way in which we designed those lower pieces, when it came time to place them into the greater design, we ran into a conflict that could only be resolved by completely redesigning the rules that we had already implemented. The point at which we despaired a bit was when we had designed mathematical and boolean expression rules, variable declarations, and control flow when we realized that the only way in which we could incorporate both control flow and variable handling was a complete overhaul of the architecture we built in Java to handle those variables as well as the grammar itself. We simply didn't have time to do this, so we were faced with a terrible choice: either include variable handling or more advanced control flow. Since variables are absolutely integral to our (or any) language and it was possible to accomplish the initial goal of our language without the features in control that we wanted, we made the tough decision to cut those features.

This decision was a tough one, but it really taught us that the most important thing to keep in mind when we, as programmers and designers, have to face choices about the design of a project is the final goal of that project. A language built to support the creation and analyzation of music would be useless without the ability to represent notes and chords, and we needed to remember to always keep that in mind. That said, we feel our project turned out wonderfully, even with the cuts that we had to make, and it taught us all that despite the difficulties that come about with any project, working towards a solid final goal will result in something great.

10. Appendix

10.1 Cheat Sheet (summary of functions, declarations, etc)

Object types:

Type	Description
Note	Represents a note. It is comprised of a pitch and a duration (an integer value that represents the length of the note in eighth notes). It also stores variables that represent the instrument the note will be played by, which will be, by default "acoustic grand", “black”, and “default” note shape. Optionally, it can also store variables for color, shape, size, and position, which will be used to represent the note graphically. All these variables will be initialized to null, or possibly some predetermined default, and they can be changed by the programmer. **To declare**: Note exampleNote = new Note(C, 4, 2); //This will be a quarter note that plays the pitch defined by C on octave 4 with duration 2
Chord	Represents any number (1 or greater) of notes played simultaneously, and is considered by our language to be the same as a note when placed into a sequence. It is composed of a list of Notes. **To declare:** Chord exampleChord = new Chord(exampleNote, example1Note, example2Note);
Sequence	Represents any number of Notes, Chords, and Sequences played in succession. It is declared empty and can be built up with the “<<” operator. **To declare:** Sequence exampleSequence = new Sequence();

Operators:

Operator	Description

var = expr
 	
return: type of ‘expr’
	Simple assignment operator

Sequence << Chord Sequence << Note Sequence << Sequence	Adds a Chord to an existing Sequence **Ex:** sequence1 << chord1

Sequence << Chord1 + Note + Chord3 + Sequence2 **return: Sequence**	sequence1 << chord1 + note2 + chord3 + sequence2

Note^int
return: nothing
	Changes the duration of the note to int
	Ex: note1^2 ; //note2 is now a half-note
	note2^8 ; //note2 is now an eight-note

Note < Color('color') **return: nothing**	Assigns a color to a Note. **Ex:** note1 < Color('green') ;
Note < Shape('shape') **return: nothing**	Assigns a Shape to a Note. **Ex:** note1 < Shape('triangle')
Note < Instrument('instrument') **return: nothing**	Assigns an Instrument to a Note. **Ex**: note1 < Instrument('guitar')

Built-in Functions:

Function	Description

Sequence(int)
 	
return: Chord
	Returns Chord at index int
	
	Ex: sequence1(2) ;

Sequence<int1, int2> **return: Note**	Returns int2th Note from Chord at int1 **Ex:** sequence1<1, 2> ; //returns 2nd note from at position 1
Sequence[int1:int2] **return: Sequence**	Returns new Sequence consisting of all chords from position int1 to position int2. (int2 can be "end" to allow access from position int1 to end of the Sequence.) **Ex:** sequence1{2: 6} ; // returns sequence of Chords2-Chords6
Sequence[int1, int2, int3...]	Returns new Sequence consisting of all

return: Sequence	chords in position int1, position int2 and position int3. **Ex:** sequence1{1,3,5} ; //returns sequence of Chord1+Chord3+Chord5 ;
[Sequence1, Sequence2] **return: Sequence**	Concatenates two Sequences **Ex:** sequence1 = {sequence2, sequence3} ;

foreach(Sequences)<Color('color');
 	
foreach(Sequence s)<Shape('shape');
 	
foreach(Sequences)<Instrument('instrument');
	This control is the most important in our language. It is implemented as a function. It iterates through all elements of a Sequence. It iterates through every single note in a sequence, including those inside chords and applies an attribute to them.
	
	ex: foreach (exampleSequence)<Color('green');

display(Sequence sequence1, "header")	Writes lilypond code to .ly and .html files to display them. Takes in a header string.

10.2 Subversion Log

r257	adding each person's code	Today (34 minutes ago)	ire.alvarado@gmail.com
r256	adding each person's code	Today (34 minutes ago)	ire.alvarado@gmail.com
r255	hello world file	Today (69 minutes ago)	ire.alvarado@gmail.com
r254	updating final versions. Taking out unused code	Today (70 minutes ago)	ire.alvarado@gmail.com
r253	deleting control	Today (3 hours ago)	ire.alvarado@gmail.com
r252	adding twinklef ile	Today (5 hours ago)	ire.alvarado@gmail.com
r251	adding twinkle twinkle	Today (5 hours ago)	ire.alvarado@gmail.com
r250	adding a test file	Today (5 hours ago)	ire.alvarado@gmail.com
r249	adding a repository for test files	Today (6 hours ago)	ire.alvarado@gmail.com
r248	adding details of sections	Today (6 hours ago)	ire.alvarado@gmail.com
r247	removing some colors that were not present	Today (9 hours ago)	ire.alvarado@gmail.com

r246	added an underscore to the ID token, allowing underscores to be apart of an ID (some place after the first letter)	Today (10 hours ago)	dunn.jonw@gmail.com
r245	[No log message]	Yesterday (19 hours ago)	rmichaelboyle@gmail.com
r244	added re-initializechord	Yesterday (19 hours ago)	rmichaelboyle@gmail.com
r243	added re-initializechord	Yesterday (19 hours ago)	rmichaelboyle@gmail.com
r242	updated CHANGE_DURATION method	Yesterday (19 hours ago)	dunn.jonw@gmail.com
r241	adding different test files for each	Yesterday (20 hours ago)	ire.alvarado@gmail.com
r240	trying to get javadoc to execute properly	Yesterday (20 hours ago)	ire.alvarado@gmail.com
r239	making it executable	Yesterday (20 hours ago)	ire.alvarado@gmail.com
r238	making it executable	Yesterday (20 hours ago)	ire.alvarado@gmail.com
r237	javadoc update	Yesterday (20 hours ago)	ire.alvarado@gmail.com
r236	general formatting	Yesterday (20 hours ago)	ire.alvarado@gmail.com
r235	general formatting	Yesterday (20 hours ago)	ire.alvarado@gmail.com
r234	merging	Yesterday (20 hours ago)	ire.alvarado@gmail.com
r233	general formatting	Yesterday (20 hours ago)	ire.alvarado@gmail.com
r232	adding javadoc	Yesterday (20 hours ago)	rmichaelboyle@gmail.com
r231	assigning stuff	Yesterday (21 hours ago)	ire.alvarado@gmail.com
r230	assigning parts	Yesterday (21 hours ago)	ire.alvarado@gmail.com
r229	final report skeleton	Yesterday (21 hours ago)	ire.alvarado@gmail.com
r228	small javadoc change	Yesterday (22 hours ago)	ire.alvarado@gmail.com
r227	javadoc executable	Yesterday (22 hours ago)	ire.alvarado@gmail.com

r226	changing property of javadoc	Yesterday (22 hours ago)	ire.alvarado@gmail.com
r225	javadoc for lilypond	Yesterday (22 hours ago)	ire.alvarado@gmail.com
r224	delete	Yesterday (22 hours ago)	ire.alvarado@gmail.com
r223	javadoc!	Yesterday (22 hours ago)	ire.alvarado@gmail.com
r222	adding javadoc code and correct formatting	Yesterday (22 hours ago)	ire.alvarado@gmail.com
r221	Added javadoc garbage	Yesterday (22 hours ago)	rmichaelboyle@gmail.com
r220	formatting	Yesterday (30 hours ago)	ire.alvarado@gmail.com
r219	public garbage	Yesterday (30 hours ago)	rmichaelboyle@gmail.com
r218	merging	Yesterday (30 hours ago)	ire.alvarado@gmail.com
r217	merging	Yesterday (30 hours ago)	ire.alvarado@gmail.com
r216	adding code to catch general errors (not dealt with in all other functions)	Yesterday (30 hours ago)	ire.alvarado@gmail.com
r215	foreach	Yesterday (30 hours ago)	rmichaelboyle@gmail.com
r214	foreach	Yesterday (30 hours ago)	rmichaelboyle@gmail.com
r213	foreach changing	Yesterday (31 hours ago)	ire.alvarado@gmail.com
r212	changed built in function syntax	Yesterday (31 hours ago)	ire.alvarado@gmail.com
r211	fixed minor error	Yesterday (31 hours ago)	rmichaelboyle@gmail.com
r210	general update	Yesterday (31 hours ago)	ire.alvarado@gmail.com
r209	merging errors	Yesterday (31 hours ago)	ire.alvarado@gmail.com
r208	adding math expressions grammar by taylor	Yesterday (31 hours ago)	ire.alvarado@gmail.com
r207	added initialize	Yesterday (32 hours ago)	rmichaelboyle@gmail.com
r206	Added more functions	Yesterday (32 hours	rmichaelboyle@gmail.com

		ago)	
r205	Added more error checking, with Jonathans code	Yesterday (32 hours ago)	rmichaelboyle@gmail.com
r204	merging and adding errors	Yesterday (34 hours ago)	rmichaelboyle@gmail.com
r203	adding final project sections from Aho's webiste	May 5 (45 hours ago)	ire.alvarado@gmail.com
r202	a todo list	May 5 (47 hours ago)	ire.alvarado@gmail.com
r201	test file, plays twinkle twinkle, with different instruments, shapes, colors	May 5 (47 hours ago)	ire.alvarado@gmail.com
r200	adding code to actually output the html file -also deletes folder "out" before calling lilypond-book	May 5 (47 hours ago)	ire.alvarado@gmail.com
r199	correcting some code for writing lilypond .html and .ly files -had not output instrument and tempo correctly	May 5 (47 hours ago)	ire.alvarado@gmail.com
r198	correcting some code for writing lilypond .html and .ly files -had not output instrument and tempo correctly	May 5 (47 hours ago)	ire.alvarado@gmail.com
r197	deleting extra stuff we don't need and correcting some warnings	May 5 (2 days ago)	ire.alvarado@gmail.com
r196	deleting	May 5 (2 days ago)	ire.alvarado@gmail.com
r195	deleting	May 5 (2 days ago)	ire.alvarado@gmail.com
r194	general update of automatic files to match those on repository	May 5 (2 days ago)	ire.alvarado@gmail.com
r193	general update of automatic files to match those on repository	May 5 (2 days ago)	ire.alvarado@gmail.com
r192	general update of automatic files to match those on repository	May 5 (2 days ago)	ire.alvarado@gmail.com
r191	deleting	May 5 (2 days ago)	ire.alvarado@gmail.com
r190	deleting	May 5 (2 days ago)	ire.alvarado@gmail.com
r189	deleting colors	May 5 (2 days ago)	ire.alvarado@gmail.com

r188	adding code for IF: grammar	May 5 (2 days ago)	ire.alvarado@gmail.com
r187	why?	May 5 (2 days ago)	ire.alvarado@gmail.com
r186	commiting changes to functions, working functions to add all types of sequences with the << operator -also merging with jonathan error checking	May 5 (2 days ago)	ire.alvarado@gmail.com
r185	commiting changes to functions, working functions to add all types of sequences with the << operator -also merging with jonathan error checking	May 5 (2 days ago)	ire.alvarado@gmail.com
r184	commiting changes to functions, working functions to add all types of sequences with the << operator -also merging with jonathan error checking	May 5 (2 days ago)	ire.alvarado@gmail.com
r183	merging with richard/jonathan -changing CONCAT_SEQ	May 5 (2 days ago)	ire.alvarado@gmail.com
r182	changed concat_seq function	May 5 (2 days ago)	ire.alvarado@gmail.com
r181	changing the concatenate function	May 5 (2 days ago)	ire.alvarado@gmail.com
r180	garbage	May 5 (2 days ago)	rmichaelboyle@gmail.com
r179	blank note constructor	May 5 (2 days ago)	rmichaelboyle@gmail.com
r178	added Jonathans code.	May 5 (2 days ago)	rmichaelboyle@gmail.com
r177	adding error checking for functions	May 5 (2 days ago)	rmichaelboyle@gmail.com
r176	merge	May 5 (2 days ago)	rmichaelboyle@gmail.com
r175	merging with richard	May 5 (2 days ago)	ire.alvarado@gmail.com
r174	merge, updated stuffs	May 5 (2 days ago)	rmichaelboyle@gmail.com
r173	merge	May 5 (2 days ago)	rmichaelboyle@gmail.com
r172	changing the tag 1 in NOTE ASSIGN rule to compile on my machine	May 5 (2 days ago)	ire.alvarado@gmail.com
r171	adding code to correct some functions, and print accidentals	May 5 (2 days ago)	ire.alvarado@gmail.com
r170	adding code to correct {sequence1, sequence2} function	May 5 (2 days ago)	ire.alvarado@gmail.com
r169	can check for # b	May 5 (2 days ago)	ire.alvarado@gmail.com

r168	merging with richard	May 5 (2 days ago)	ire.alvarado@gmail.com
r167	added check to first Note method	May 4 (3 days ago)	dunn.jonw@gmail.com
r166	added alreadyExists method to Error	May 4 (3 days ago)	dunn.jonw@gmail.com
r165	added	May 4 (3 days ago)	rmichaelboyle@gmail.com
r164	merge	May 4 (3 days ago)	rmichaelboyle@gmail.com
r163	added new note assignment	May 4 (3 days ago)	rmichaelboyle@gmail.com
r162	merging	May 4 (3 days ago)	tro2102
r161	merging	May 4 (3 days ago)	tro2102
r160	merging	May 4 (3 days ago)	tro2102
r159	merging taylor	May 4 (3 days ago)	tro2102
r158	adding import java.lang.Error so it does not confuse error classes	May 4 (3 days ago)	ire.alvarado@gmail.com
r157	merging jonathan/irene	May 4 (3 days ago)	ire.alvarado@gmail.com
r156	merging richard irene	May 4 (3 days ago)	ire.alvarado@gmail.com
r155	merging jonathan, irene	May 4 (3 days ago)	ire.alvarado@gmail.com
r154	merging richard and jonathan	May 4 (3 days ago)	dunn.jonw@gmail.com
r153	adding an error class	May 4 (3 days ago)	dunn.jonw@gmail.com
r152	corrected string text to parse '	May 4 (3 days ago)	dunn.jonw@gmail.com
r151	added some stuff	May 4 (3 days ago)	rmichaelboyle@gmail.com
r150	added some stuff	May 4 (3 days ago)	rmichaelboyle@gmail.com
r149	adding code to not add a Null chord to subsequence	May 4 (3 days ago)	ire.alvarado@gmail.com
r148	renaming sequenceFrom2 to sequenceFrom	May 4 (3 days ago)	ire.alvarado@gmail.com
r147	adding the Sequence << Note + Chord operation	May 4 (3 days ago)	ire.alvarado@gmail.com
r146	added syso to a bunch of crap	May 4 (3 days ago)	rmichaelboyle@gmail.com
r145	added another subsequence function	May 4 (3 days ago)	rmichaelboyle@gmail.com
r144	merge	May 4 (3 days ago)	rmichaelboyle@gmail.com
r143	added subsequence stuffs	May 4 (3 days ago)	rmichaelboyle@gmail.com
r142	added subsequence stuffs	May 4 (3 days ago)	rmichaelboyle@gmail.com
r141	deleting code to complicate Map, easier to check type with instanceof	May 4 (3 days ago)	ire.alvarado@gmail.com
r140	deleting extra code	May 4 (3 days ago)	ire.alvarado@gmail.com
r139	deleting	May 4 (3 days ago)	rmichaelboyle@gmail.com
r138	deleting files --> moving to src	May 3 (4 days ago)	ire.alvarado@gmail.com

r137	adding a test file and shell script	May 3 (4 days ago)	ire.alvarado@gmail.com
r136	deleting	May 3 (4 days ago)	ire.alvarado@gmail.com
r135	deleting	May 3 (4 days ago)	ire.alvarado@gmail.com
r134	automatic files generated after changes	May 3 (4 days ago)	ire.alvarado@gmail.com
r133	deleting	May 3 (4 days ago)	ire.alvarado@gmail.com
r132	changing name of token CHANGE_DURATION	May 3 (4 days ago)	ire.alvarado@gmail.com
r131	deleting	May 3 (4 days ago)	ire.alvarado@gmail.com
r130	deleting	May 3 (4 days ago)	ire.alvarado@gmail.com
r129	adding code to call shell script	May 3 (4 days ago)	ire.alvarado@gmail.com
r128	accepting Richard's changes	May 3 (4 days ago)	ire.alvarado@gmail.com
r127	accepting Richard's changes	May 3 (4 days ago)	ire.alvarado@gmail.com
r126	script to copy into mus directory and executy lilypond and lilypond-book	May 3 (4 days ago)	ire.alvarado@gmail.com
r125	update	May 3 (4 days ago)	rmichaelboyle@gmail.com
r124	Added various functions assignment, subsequence, get chords etc	May 3 (4 days ago)	rmichaelboyle@gmail.com
r123	Added various functions assignment, subsequence, get chords etc	May 3 (4 days ago)	rmichaelboyle@gmail.com
r122	merge	May 3 (4 days ago)	rmichaelboyle@gmail.com
r121	mergedddd	May 3 (4 days ago)	rmichaelboyle@gmail.com
r120	adding parse for DISPLAY function	May 3 (4 days ago)	ire.alvarado@gmail.com
r119	adding parse for DISPLAY function	May 3 (4 days ago)	ire.alvarado@gmail.com
r118	adding parse for DISPLAY function	May 3 (4 days ago)	ire.alvarado@gmail.com
r117	Changed variable HashMap to contain variable type	May 3 (4 days ago)	ire.alvarado@gmail.com
r116	updated with various functions	May 3 (4 days ago)	rmichaelboyle@gmail.com
r115	[No log message]	May 3 (4 days ago)	dunn.jonw@gmail.com
r114	added \r to the WS token.	May 3 (4 days ago)	dunn.jonw@gmail.com
r113	merge	May 3 (4 days ago)	rmichaelboyle@gmail.com
r112	merge	May 3 (4 days ago)	rmichaelboyle@gmail.com
r111	correcting code CONCATENATE_SEQUENCE to	May 3 (4 days ago)	ire.alvarado@gmail.com

	return a sequence		
r110	removing "static" out of some functions	May 3 (4 days ago)	ire.alvarado@gmail.com
r109	adding code to clone a Sequence	May 3 (4 days ago)	ire.alvarado@gmail.com
r108	adding code to print a Note	May 3 (4 days ago)	ire.alvarado@gmail.com
r107	adding code to print a Chord	May 3 (4 days ago)	ire.alvarado@gmail.com
r106	merge	May 3 (4 days ago)	rmichaelboyle@gmail.com
r105		May 3 (4 days ago)	rmichaelboyle@gmail.com
r104	adding code to correctly print out Note and Chord	May 3 (4 days ago)	ire.alvarado@gmail.com
r103	adding code to correctly print out Note and Chord	May 3 (4 days ago)	ire.alvarado@gmail.com
r102	adding tags to Reference for check color, string, instrument so it does not spit an error when first initializing Note	May 3 (4 days ago)	ire.alvarado@gmail.com
r101	adding tags to Reference for check color, string, instrument so it does not spit an error when first initializing Note	May 3 (4 days ago)	ire.alvarado@gmail.com
r100	adding code to assign color, shape, instrument, etc	May 3 (4 days ago)	rmichaelboyle@gmail.com
r99	adding code to assign color, shape, instrument, etc	May 3 (4 days ago)	rmichaelboyle@gmail.com
r98	adding Shape, Instrument, LessThan tokens	May 3 (4 days ago)	rmichaelboyle@gmail.com
r97	adding code to clone objects that are assigned	May 3 (4 days ago)	ire.alvarado@gmail.com
r96	Adding code to 1) add Chord to sequence 2) clone the objects that are assigned	May 3 (4 days ago)	ire.alvarado@gmail.com
r95	Adding type for a basic Sequence	29-Apr-11	ire.alvarado@gmail.com
r94	deleting Class Variable, it has been moved to lexer>ParseTree.java	29-Apr-11	ire.alvarado@gmail.com
r93	Adding a new class ParserTree to execute actions between CUP parser and java code	29-Apr-11	ire.alvarado@gmail.com

r92	adding terminal FOREACH to Cup file	29-Apr-11	ire.alvarado@gmail.com
r91	adding parsing code for Chord type	29-Apr-11	ire.alvarado@gmail.com
r90	added add note to chord in chord class	29-Apr-11	rmichaelboyle@gmail.com
r89	added add note to chord functionality	29-Apr-11	rmichaelboyle@gmail.com
r88	adding basic grammar tree	29-Apr-11	ire.alvarado@gmail.com
r87	added code for "foreach"	29-Apr-11	tro2102
r86	adding sequence terminal	29-Apr-11	ire.alvarado@gmail.com
r85	example test file	29-Apr-11	ire.alvarado@gmail.com
r84	correcting Note and Chord grammar to take "new" terminal	29-Apr-11	ire.alvarado@gmail.com
r83	adding code for new terminal	29-Apr-11	tro2102
r82	adding new terminal	29-Apr-11	tro2102
r81	adding code to check variables. Variable table	29-Apr-11	ire.alvarado@gmail.com
r80	adding grammar for chords	29-Apr-11	ire.alvarado@gmail.com
r79	adding grammar to check for code	29-Apr-11	ire.alvarado@gmail.com
r78	changing file to show two lexers output	28-Apr-11	dunn.jonw@gmail.com
r77	changing lilypond main	24-Apr-11	ire.alvarado@gmail.com
r76	changing Reference to use the ojbect constructor	24-Apr-11	ire.alvarado@gmail.com
r75	deleting extraText	24-Apr-11	ire.alvarado@gmail.com
r74	additional, extra files not used	24-Apr-11	ire.alvarado@gmail.com
r73	adding a lilypond package to build kernel	24-Apr-11	ire.alvarado@gmail.com
r72	adding a lilypond package to build kernel	24-Apr-11	ire.alvarado@gmail.com
r71	small working kernel parser	24-Apr-11	ire.alvarado@gmail.com
r70	deleting extra folder	24-Apr-11	ire.alvarado@gmail.com
r69	updating working cup file. Semantic analyzer	24-Apr-11	ire.alvarado@gmail.com
r68	adding grammar tree	24-Apr-11	ire.alvarado@gmail.com
r67	adding farbound's cupf file	24-Apr-11	ire.alvarado@gmail.com
r66	Updating everything	24-Apr-11	ire.alvarado@gmail.com
r65	WORKING LEXER!!	24-Apr-11	ire.alvarado@gmail.com

r64	included the cup file	24-Apr-11	farbound
r63	commiting yo.	24-Apr-11	rmichaelboyle@gmail.com
r62	testing again, last try to get lexical	24-Apr-11	ire.alvarado@gmail.com
r61	adding a test	24-Apr-11	ire.alvarado@gmail.com
r60	trying to get lexical anlyzer to work	24-Apr-11	ire.alvarado@gmail.com
r59	completed lexical anlyzer, with updates	24-Apr-11	ire.alvarado@gmail.com
r58	new lexical anlyzer	24-Apr-11	ire.alvarado@gmail.com
r57	making changes to symbol table	24-Apr-11	ire.alvarado@gmail.com
r56	yo	24-Apr-11	rmichaelboyle@gmail.com
r55	Adding files to do lexical analysis	24-Apr-11	ire.alvarado@gmail.com
r54	lex	24-Apr-11	rmichaelboyle@gmail.com
r53	updating Sequence with concatenate function	24-Apr-11	ire.alvarado@gmail.com
r52	new main	24-Apr-11	ire.alvarado@gmail.com
r51	updating note and chord	24-Apr-11	ire.alvarado@gmail.com
r50	adding sequence functions	24-Apr-11	rmichaelboyle@gmail.com
r49	accepting changes	24-Apr-11	rmichaelboyle@gmail.com
r48	accepting changes	24-Apr-11	rmichaelboyle@gmail.com
r47	adding new functions to sequences	24-Apr-11	ire.alvarado@gmail.com
r46	updated convert	17-Apr-11	rmichaelboyle@gmail.com
r45	adding octave checking	17-Apr-11	ire.alvarado@gmail.com
r44	updated lilypond convert	17-Apr-11	rmichaelboyle@gmail.com
r43	Updating to check for octaves and output correct tickings	17-Apr-11	ire.alvarado@gmail.com
r42	changed write function	17-Apr-11	rmichaelboyle@gmail.com
r41	adding some code	17-Apr-11	ire.alvarado@gmail.com
r40	updated octave symbols in note	17-Apr-11	rmichaelboyle@gmail.com
r39	adding more code to check for pitch symbols	17-Apr-11	ire.alvarado@gmail.com
r38	Adding code to check for pitch	17-Apr-11	ire.alvarado@gmail.com
r37	Adding code for masterSequence writing out	17-Apr-11	ire.alvarado@gmail.com
r36	adding a function o output main sequence	17-Apr-11	ire.alvarado@gmail.com

r35	updated list	17-Apr-11	rmichaelboyle@gmail.com
r34	adding convertNote to Chord class	17-Apr-11	ire.alvarado@gmail.com
r33	accepting test	17-Apr-11	ire.alvarado@gmail.com
r32	updating Sequence	17-Apr-11	rmichaelboyle@gmail.com
r31	some changes	17-Apr-11	rmichaelboyle@gmail.com
r30	changing name of main from 'test' to 'LilypondConvert'	17-Apr-11	ire.alvarado@gmail.com
r29	added duration function	17-Apr-11	rmichaelboyle@gmail.com
r28	adding code to test for errors in Reference class	17-Apr-11	ire.alvarado@gmail.com
r27	adding error checking for reference tables	17-Apr-11	ire.alvarado@gmail.com
r26	new lex analyzer.	17-Apr-11	dunn.jonw@gmail.com
r25	deleting mymusicapp.java its in the extra package	16-Apr-11	ire.alvarado@gmail.com
r24	accepting change	16-Apr-11	rmichaelboyle@gmail.com
r23	new test.java	16-Apr-11	rmichaelboyle@gmail.com
r22	[No log message]	16-Apr-11	ire.alvarado@gmail.com
r21	test.java with added html output support	16-Apr-11	rmichaelboyle@gmail.com
r20	changed name of file	16-Apr-11	ire.alvarado@gmail.com
r19	more extra files, realted to Jfugue and java MIDI	16-Apr-11	ire.alvarado@gmail.com
r18	extra files, not really using	16-Apr-11	ire.alvarado@gmail.com
r17	test file for creating lilypond source	16-Apr-11	rmichaelboyle@gmail.com
r16	Fixed chord to properly output ly code	16-Apr-11	rmichaelboyle@gmail.com
r15	First basic Chord class -RMB	16-Apr-11	rmichaelboyle@gmail.com
r14	First basic Note class -RMB	16-Apr-11	rmichaelboyle@gmail.com
r13	general update	16-Apr-11	ire.alvarado@gmail.com
r12	adding reference files with tables for instruments, pitches, etc.	16-Apr-11	ire.alvarado@gmail.com
r11	adding code to output XMLmusic file	10-Apr-11	ire.alvarado@gmail.com
r10	changes to subfolders	10-Apr-11	ire.alvarado@gmail.com
r9	example parser and renderer	10-Apr-11	ire.alvarado@gmail.com
r8	[No log message]	10-Apr-11	rmichaelboyle@gmail.com
r7	adding a jfugue package library	10-Apr-11	ire.alvarado@gmail.com

r6	Testing eclipse google code	5-Apr-11	tro2102
r5	testing for irene	5-Apr-11	ire.alvarado@gmail.com
r4	this is farbound's test folder	4-Apr-11	farbound
r3	[No log message]	4-Apr-11	rmichaelboyle@gmail.com
r2	Test	4-Apr-11	tro2102@gmail.com
r1	Initial directory structure.	4-Apr-11	---

10.3 Code

10.3.a Front End: package compiler

Created code:

- Lexical Analyzer: "LexicalAnalyzer.lex"
- Parser: "Parser.cup"
- Parse Tree: "ParseTree.java"
    - Interface between front end and back end
- Error class: "Error.java"
- Main class: "Main.java"
    - Creates a lexical analyzer and parser and runs it through .mus file

Code generated automatically:

- "Yylex": generated from the jlex file
- "ParserCup.java": generated from the CUP file
- "ParserSym.java": generated from the CUP file, it is the symbol table.

10.3.b Back End: package lilypond

- Note class: "Note.java"
- Chord class: "Chord.java"
- Sequence class: "Sequence.java"
- Repository to store defaults: "Reference.java"
- Class to write correct output to lilypond: "LilypondConvert.java"

10.3.c Other files

- "lilypond.sh": Script file to call on 'lilypond' and 'lilypond-book' programs to convert a .ly file to a .midi file and a .html file without graphics to a .html file with graphics

10.3.d Javadoc

Can be found here:

http://mus.googlecode.com/svn/trunk/javadoc/index.html

10.3.e Code not used because of time limitations

Code that allows for integer variables, Booleans, and math expressions

10.3.f Test files

We made several different kinds of test files, the output of which can be found online at:

http://code.google.com/p/mus/source/browse/#svn%2Ftrunk%2Fsrc%2Ftestfiles

- Checking attributes:
    - NoteAttributeColor.mus
    - NoteAttributeShape.mus
    - NoteAttributeInstrument.mus
    - NoteAttributePitch.mus
- Checking for compiler error checking
    - BuiltinFuncitons.mus
    - DeclarationErrors.mus
- General files to output music:
    - Twinkle.mus
    - Pachelbel.Canon.mus
    - Aho.mus

Front-End

```
//*Jonathan*//

package lexer;

import java.io.FileReader;
import java.io.IOException;
import java_cup.runtime.*;
import java.lang.Error ;


class Utility
{
    public static void check(boolean expr)
    {
        if (false == expr)
            throw (new java.lang.Error("Error: Assertion failed."));
    }

    private static final String[] errorMessage = {
            "Error: Unmatched end-of-comment punctuation.",
            "Error: Unmatched start-of-comment punctuation.",
            "Error: Unclosed string.", "Error: Illegal
character." };
    public static final int E_ENDCOMMENT = 0;
    public static final int E_STARTCOMMENT = 1;
    public static final int E_UNCLOSEDSTR = 2;
    public static final int E_UNMATCHED = 3;

    public static void error(int code)
    {
        System.out.println(errorMessage[code]);
    }
}



%%
%{
  private int comment_counter = 0;


```

```
 class Symbol{
 Symbol(int consti, String text, Object value, int charBegin, int charEnd){
   this.value = value;
   this.text = text;
   this.charBegin = charBegin;
   this.charEnd = charEnd;
   this.consti = consti;
 }

   public String toString(){
      return "Token text:" + text + ". Value is:" + value + ".";
   }

   private int charBegin, charEnd;
   String text;
   Object value ;
   int consti ;

   public java_cup.runtime.Symbol returnSymbol()
   {
    java_cup.runtime.Symbol x = new java_cup.runtime.Symbol(consti, charBegin, charEnd, value ) ;
    return x ;
   }
}


%}

%line
%state COMMENT, COMMENTLINE
%cupsym ParserCup
%cup
%eofval{
   return new java_cup.runtime.Symbol(ParserSym.EOF);
%eofval}
%char
WS = [ \n\r\t\b\012]
LETTER = [A-Za-z]
DIGIT = [0-9]
```

```
ID = {LETTER}({LETTER}|{DIGIT}|\#|_)*
NUMBER = {DIGIT}+(\.{DIGIT}*)?(E[-+]?{DIGIT}+)?
STRING_TEXT = (\\\"|[^\n\"\']|\\{WS}+\\)*
COMMENT_TEXT = ([^/*\n]|[^*\n]"/"[^*\n]|[^/\n]"*"[^/\n]|"*"[^/
\n]|"/"[^*\n])*
%%




<YYINITIAL> "new" { return new Symbol(ParserSym.NEW, yytext(), null,
yychar, yychar+3).returnSymbol();}
<YYINITIAL> "Color" { return new Symbol(ParserSym.COLOR, yytext(),
null, yychar, yychar+5).returnSymbol();}
<YYINITIAL> "Instrument" { return new Symbol(ParserSym.INSTRUMENT,
yytext(), null, yychar, yychar+10).returnSymbol();}
<YYINITIAL> "Shape" {return new Symbol(ParserSym.SHAPE, yytext(),
null, yychar, yychar+5).returnSymbol();}
<YYINITIAL> "Note" { return new Symbol(ParserSym.NOTE, yytext(),
null, yychar, yychar+4).returnSymbol();}
<YYINITIAL> "Chord" { return new Symbol(ParserSym.CHORD, yytext(),
null, yychar, yychar+5).returnSymbol();}
<YYINITIAL> "Sequence" { return new Symbol(ParserSym.SEQUENCE,
yytext(), null, yychar, yychar+8).returnSymbol();}
<YYINITIAL> "foreach" { return new Symbol(ParserSym.FOREACH, yytext
(), null, yychar, yychar+7).returnSymbol();}
<YYINITIAL> "," { return new Symbol(ParserSym.COMMA, yytext(), null,
yychar, yychar+1).returnSymbol();}
<YYINITIAL> ";" { return new Symbol(ParserSym.SEMICOLON, yytext(),
null, yychar, yychar+4).returnSymbol();}
<YYINITIAL> "(" { return new Symbol(ParserSym.LPAREN, yytext(),
null, yychar, yychar+1).returnSymbol();}
<YYINITIAL> ")" { return new Symbol(ParserSym.RPAREN, yytext(),
null, yychar, yychar+1).returnSymbol();}
<YYINITIAL> "[" { return new Symbol(ParserSym.LBRACK, yytext(),
null, yychar, yychar+1).returnSymbol();}
<YYINITIAL> "]" { return new Symbol(ParserSym.RBRACK, yytext(),
null, yychar, yychar+1).returnSymbol();}
<YYINITIAL> "+" { return new Symbol(ParserSym.PLUS, yytext(), null,
yychar, yychar+1).returnSymbol();}
<YYINITIAL> "=" { return new Symbol(ParserSym.ASSIGN, yytext(),
null, yychar, yychar+1).returnSymbol();}
<YYINITIAL> ">" { return new Symbol(ParserSym.GTHAN, yytext(), null,
yychar, yychar+1).returnSymbol();}
<YYINITIAL> "<<" { return new Symbol(ParserSym.ADD_TO_SEQUENCE,
yytext(), null, yychar, yychar+2).returnSymbol();}
<YYINITIAL> "^" { return new Symbol(ParserSym.CHANGE_DURATION,
yytext(), null, yychar, yychar+1).returnSymbol();}
<YYINITIAL> "<" { return new Symbol(ParserSym.LTHAN, yytext(), null,
yychar, yychar+1).returnSymbol();}
<YYINITIAL> "display" { return new Symbol(ParserSym.DISPLAY, yytext
(), null, yychar, yychar+7).returnSymbol();}
<YYINITIAL> ":" { return new Symbol(ParserSym.COLON, yytext(), null,
yychar, yychar+1).returnSymbol();}


<YYINITIAL, COMMENT> {WS} { }
<YYINITIAL> "//" { yybegin(COMMENTLINE); }
<COMMENTLINE> [^\n] { }
<COMMENTLINE> [\n] { yybegin(YYINITIAL); }

<YYINITIAL> "/*" { yybegin(COMMENT); comment_counter++; }
<COMMENT> "/*" { comment_counter++; }
<COMMENT> "*/" {
     comment_counter--;
     Utility.check(comment_counter >= 0);
     if (comment_counter == 0) yybegin(YYINITIAL);
}
<COMMENT> {COMMENT_TEXT} { }

<YYINITIAL> \"{STRING_TEXT}\" {
    String str = yytext().substring(1, yytext().length() - 1);

    Utility.check(str.length() == yytext().length() - 2);
    return (new Symbol(ParserSym.STRING_TEXT, str, str, yychar,
yychar + str.length()).returnSymbol());
}


<YYINITIAL> \"{STRING_TEXT} {
       String str = yytext().substring(1, yytext().length());

```

```
        Utility.error(Utility.E_UNCLOSEDSTR);
        Utility.check(str.length() == yytext().length() - 1);
        return (new Symbol(ParserSym.STRING_TEXT, str, str, yychar,
 yychar + str.length()).returnSymbol());
}


<YYINITIAL> \'{STRING_TEXT}\' {
        String str = yytext().substring(1, yytext().length() - 1);

        Utility.check(str.length() == yytext().length() - 2);
        return (new Symbol(ParserSym.STRING_TEXT, str, str, yychar,
 yychar + str.length()).returnSymbol());
}


<YYINITIAL> \'{STRING_TEXT} {
         String str = yytext().substring(1, yytext().length());

         Utility.error(Utility.E_UNCLOSEDSTR);
     Utility.check(str.length() == yytext().length() - 1);
         return (new Symbol(ParserSym.STRING_TEXT, str, str, yychar,
 yychar + str.length()).returnSymbol());
}


<YYINITIAL> {NUMBER} {
        return (new Symbol(ParserSym.DIGIT, yytext(), new Integer
 (yytext()), yychar, yychar + yytext().length()).returnSymbol());
}
<YYINITIAL> {ID} {
         return (new Symbol(ParserSym.ID, yytext(), yytext(), yychar,
 yychar + yytext().length()).returnSymbol());
}
<YYINITIAL, COMMENT> . {
         System.out.println("Illegal character: <" + yytext() + ">");
         Utility.error(Utility.E_UNMATCHED);
}



```

```
//*Irene &  Richard*//

package lexer;

import java_cup.runtime.*;
import lilypond.*;
import java.util.ArrayList;




action code {:

    ParseTree tree = new ParseTree() ;

 :};

parser code {:

:};

    terminal STRING_TEXT, ID ;
    terminal NEW, ASSIGN;
    terminal COLOR,SHAPE, INSTRUMENT;
    terminal NOTE, CHORD, SEQUENCE;
    terminal COMMA, SEMICOLON, COLON ;

    terminal PLUS ;
    terminal DIGIT;

    terminal LPAREN, RPAREN, LBRACK, RBRACK ;
    terminal LTHAN, GTHAN ;

    terminal ADD_TO_SEQUENCE, CHANGE_DURATION, FOREACH, DISPLAY ;

non terminal    expr_list, expr_part ;
non terminal    note, chord, sequence ;
non terminal    assign, var ;
non terminal    add_to_sequence, concat_seq, display,
change_duration, chord_in_seq, note_in_seq, sub_seq ;
non terminal    list_variables, list_notes, list_chords ;

non terminal    foreach ;

non terminal    object_types, functions ;

precedence left PLUS;

start with expr_list ;

expr_list ::= expr_part expr_list | expr_part ;

expr_part ::=  functions | object_types | assign |  var ;

object_types ::= note | chord | sequence ;
functions ::= add_to_sequence | concat_seq SEMICOLON | display |
change_duration | chord_in_seq SEMICOLON | note_in_seq SEMICOLON |
sub_seq SEMICOLON ;

//*Irene*//

note ::= NOTE ID:name ASSIGN NEW NOTE LPAREN ID:pitch COMMA
DIGIT:octave COMMA DIGIT:duration RPAREN SEMICOLON
    {:
            tree.NOTE((String) name, (String) pitch, (Integer)
octave, (Integer) duration) ;
    :}
    |NOTE ID:name SEMICOLON
    {:
        tree.NOTE((String) name);
    :}
    |ID:name ASSIGN NEW NOTE LPAREN ID:pitch COMMA DIGIT:octave
COMMA DIGIT:duration RPAREN SEMICOLON
    {:
        tree.INITIALIZE_NOTE((String) name, (String) pitch,
(Integer) octave, (Integer) duration) ;
    :}
    ;

list_notes ::= ID:name1 COMMA list_notes:name2
    {:
        RESULT = tree.MULT_NOTES((String) name1, (Chord) name2) ;
    :}
    |
    ID:name
    {:
        RESULT = tree.ONE_NOTE((String) name) ;
    :}
    ;

chord ::= CHORD ID:name ASSIGN NEW CHORD LPAREN list_notes:notes RPAREN SEMICOLON
    {:
        tree.CHORD((String) name, (Chord) notes) ;
    :}
    |CHORD ID:name SEMICOLON
    {:
        tree.CHORD((String) name);
    :}
    |ID:name ASSIGN NEW CHORD LPAREN list_notes:notes RPAREN SEMICOLON
    {:
        tree.INITIALIZE_CHORD((String) name, (Chord) notes) ;
    :}
    ;

list_chords ::= DIGIT:name1 COMMA list_chords:name2
    {:
        RESULT = tree.MULT_CHORDS((Integer)name1, (String)name2);
    :}
    |DIGIT:name
    {:
        RESULT = tree.ONE_CHORDS((Integer)name);
    :}
    ;

sequence ::= SEQUENCE ID:name ASSIGN NEW SEQUENCE LPAREN RPAREN SEMICOLON
    {:
        tree.SEQUENCE((String) name) ;
    :}
    |SEQUENCE ID:name SEMICOLON
    {:
        tree.SEQ((String) name);
    :}
    ;

add_to_sequence ::= ID:sequence ADD_TO_SEQUENCE list_variables:variable_list SEMICOLON
    {:
        tree.ADD_TO_SEQUENCE((String) sequence, (ArrayList<Object>) variable_list) ;
    :}
    ;

//*Richard*//

chord_in_seq ::= ID:name1 LPAREN DIGIT:num RPAREN
    {:
        RESULT = tree.CHORD_IN_SEQ((String)name1, (Integer) num);
    :}
    ;

note_in_seq ::= ID:name1 LTHAN DIGIT:num_chord COMMA DIGIT:num_note GTHAN
    {:
        RESULT = tree.NOTE_IN_SEQ((String)name1, (Integer) num_chord, (Integer)num_note);
    :}
    ;

sub_seq ::= ID:name LBRACK DIGIT:num1 COLON DIGIT:num2 RBRACK
    {:
        RESULT = tree.SUB_SEQ((String)name,(Integer) num1,(Integer) num2);
    :}
    | ID:name LBRACK list_chords:chordlist RBRACK
    {:
        RESULT = tree.SUB_SET((String)name, (String) chordlist);
    :}
    ;

concat_seq ::= LBRACK ID:s1 COMMA ID:s2 RBRACK
```

```
    {:
        RESULT = tree.CONCAT_SEQ((String) s1, (String)s2);
    :}
    ;

//*Irene*//

list_variables ::= ID:name1 PLUS list_variables:variable_list
    {:
        RESULT = tree.MULT_VARIABLES((String) name1, (ArrayList<Object>) variable_list) ;
    :}
    |
    ID:name
    {:
        RESULT = tree.ONE_VARIABLE((String) name) ;
    :}
    |
    sub_seq:name1 PLUS list_variables:variable_list
    {:
        RESULT = tree.MULT_VARIABLE_N((Object) name1, (ArrayList<Object>) variable_list) ;

    :}
    |
    sub_seq:name
    {:
        RESULT = tree.ONE_VARIABLE_N((Object) name) ;
    :}
    |
    concat_seq:name1 PLUS list_variables:variable_list
    {:
        RESULT = tree.MULT_VARIABLE_N((Object) name1, (ArrayList<Object>) variable_list) ;
    :}
    |
    concat_seq:name
    {:
        RESULT = tree.ONE_VARIABLE_N((Object) name) ;
    :}
    |
    chord_in_seq:name1 PLUS list_variables:variable_list
    {:
        RESULT = tree.MULT_VARIABLE_N((Object) name1,
(ArrayList<Object>) variable_list) ;
    :}
    |
    chord_in_seq:name
    {:
        RESULT = tree.ONE_VARIABLE_N((Object) name) ;
    :}
    |
    note_in_seq:name1 PLUS list_variables:variable_list
    {:
        RESULT = tree.MULT_VARIABLE_N((Object) name1,
(ArrayList<Object>) variable_list) ;
    :}
    |
    note_in_seq:name
    {:
        RESULT = tree.ONE_VARIABLE_N((Object) name) ;
    :}
    ;

//*Richard*//

assign ::= ID:name LTHAN COLOR LPAREN STRING_TEXT:attribute RPAREN
SEMICOLON
    {:
        tree.ATTRIBUTE_COLOR((String) name, (String) attribute);
    :}
    |ID:name LTHAN INSTRUMENT LPAREN STRING_TEXT:attribute RPAREN
SEMICOLON
    {:
        tree.ATTRIBUTE_INSTRUMENT((String) name, (String)
attribute);
    :}
    |ID:name LTHAN SHAPE LPAREN STRING_TEXT:attribute RPAREN
SEMICOLON
    {:
        tree.ATTRIBUTE_SHAPE((String) name, (String) attribute);
    :}
    | foreach:name LTHAN COLOR LPAREN STRING_TEXT:attribute RPAREN SEMICOLON
    {:
        tree.FOREACH_COLOR((String) name,(String)attribute);
    :}
    | foreach:name  LTHAN SHAPE LPAREN STRING_TEXT:attribute RPAREN SEMICOLON
    {:
        tree.FOREACH_SHAPE((String) name,(String) attribute);
    :}
    | foreach:name  LTHAN INSTRUMENT LPAREN STRING_TEXT:attribute RPAREN SEMICOLON
    {:
        tree.FOREACH_INSTRUMENT((String) name,(String) attribute);
    :}
    ;



var ::= NOTE ID:name1 ASSIGN ID:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_NOTE((String) name1, (String)name2, true);
    :}
    | CHORD ID:name1 ASSIGN ID:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_CHORD((String) name1, (String)name2, true);
    :}
    | SEQUENCE ID:name1 ASSIGN ID:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_SEQ((String) name1, (String)name2, true);
    :}
    | ID:name1 ASSIGN ID:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR((String) name1, (String)name2);
    :}
    | CHORD ID:name1 ASSIGN chord_in_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_CHORD((String) name1, (Chord) name2, true);
    :}
    | NOTE ID:name1 ASSIGN note_in_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_NOTE((String) name1, (Note)name2, true);
    :}
    | SEQUENCE ID:name1 ASSIGN sub_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_SEQ((String) name1, (Sequence)name2, true);
    :}
    | ID:name1 ASSIGN chord_in_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_CHORD((String) name1, (Chord)name2, false);
    :}
    | ID:name1 ASSIGN note_in_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_NOTE((String) name1, (Note)name2, false);
    :}
    | ID:name1 ASSIGN sub_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_SEQ((String) name1, (Sequence)name2,false);
    :}
    | ID:name1 ASSIGN concat_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_SEQ((String) name1, (Sequence)name2,false);
    :}
    | SEQUENCE ID:name1 ASSIGN concat_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_SEQ((String) name1, (Sequence)name2, true);
    :}
    ;

//*Jonathan*//

foreach ::= FOREACH LPAREN ID:name RPAREN
    {:
            RESULT = name ;
    :}
    ;

change_duration ::= ID:name CHANGE_DURATION DIGIT:num SEMICOLON
    {:
        tree.CHANGE_DURATION((String) name, (Integer) num);
    :}
    ;
```

```


display ::= DISPLAY LPAREN ID:name COMMA STRING_TEXT:header RPAREN SEMICOLON
    {:
        tree.DISPLAY((String) name, (String) header) ;
    :}
    ;


```

```java
package compiler;

import java.io.*;

//*Irene*//

public class ParseTree {

    static Variable variables = new Variable();
    static Reference reference = new Reference()  ;
    /**
     * Constructor for Parse Tree
     * Populates the reference table
     */
    public ParseTree()
    {

        reference.populateOctaveMap() ;
    }
    /**
     * Creates new note with attributes
     * @param var_name Note variable name
     * @param pitch Pitch value as a String
     * @param octave Octave value
     * @param duration Duration value
     */
    public void NOTE(String var_name, String pitch, int octave, int
duration)
    {//declares a new note
        if (variables.contains(var_name)){ //checks to see if
variable is declared
            Object o = variables.returnVariable(var_name);
            Error.alreadyDeclared(var_name, o.getClass().getName());
        }
        //declare variable as Note
        else{
            Note note = new Note(reference.checkColor("",0),
reference.checkShape("",0), pitch,
duration,reference.checkInstrument("",0), octave);
            variables.addVariable(var_name, note) ;
            note.initialized = true;
```

```
         //System.out.println(var_name + " == " + note.printNote() + "\n");
      }
   }
   /**
    * Creates new note with no attributes, null note, called from note
    * @param name Note variable name
    */
   public void NOTE(String name)
   {//declaring a Note without initializing it ->  Note a;
      if (variables.contains(name)){ //checks to see if variable is already declared
         Object o = variables.returnVariable(name);
         Error.alreadyDeclared(name, o.getClass().getName());
      }
      else { //adds a blank Note
         Note note = new Note();
         note.initialized = false;
         variables.addVariable(name, note);
      }
   }
   /**
    * Re-initializes note that was already declared, called from note
    * @param var_name Note variable name
    * @param pitch Pitch String value
    * @param octave Octave value
    * @param duration Duration value
    */
   public void INITIALIZE_NOTE(String var_name, String pitch, int octave, int duration)
   {//reinitializes a note
      if (variables.contains(var_name)){
         Object o = variables.returnVariable(var_name);
         //if variable is a Note, reinitialize it
         if (o instanceof lilypond.Note){
            variables.removeVariable(var_name);
            Note note = new Note(reference.checkColor("",0), reference.checkShape("",0), pitch, duration,reference.checkInstrument("",0), octave);
            variables.addVariable(var_name, note);
            note.initialized = true;
            //System.out.println(var_name + " == " + note.printNote() + "\n") ;
         }
         //if variable is of another type
         else
            Error.alreadyDeclared(var_name, o.getClass().getName());
      }
      //variable has not been declared
      else
         Error.declaredOrInitialized(Error.NOT_DECLARED, var_name);
   }
   /**
    * Multiple Note function adding a new note variable to a chord, called from list_notes
    * @param note Note variable name
    * @param chord Chord variable name
    * @return returns specified Chord with new note
    */
   public Chord MULT_NOTES(String note, Chord chord)
   {//adding multiple notes to a chord. Called form list_notes
      //checking to see if variable note is a Note
      if(!variables.contains(note))
         Error.declaredOrInitialized(Error.NOT_DECLARED, note);
      else{//variable note is declared
         Object o = variables.returnVariable(note);
         if(o instanceof lilypond.Note){ //variable note is a Note
            Note n = (Note) o;
            if (chord != null){
               if (n.initialized == true) { //variable note is initialized
                  Note addNote = n.cloneNote();
                  chord.addNote(addNote);
                  return chord;
               }
               else //variable note is not initialized
                  Error.declaredOrInitialized(Error.NOT_INITIALIZED, note);
            }
            else
               Error.nullPointer();
         }
         else
            Error.isNot(note, "lilypond.Note");
      }
      return null;

   }
   /**
    * Single Note addition to a chord function, called from list_notes
    * @param note Note variable name
    * @return returns new Chord with note
    */
   public Chord ONE_NOTE(String note)
   {//Adding one note to a chord. Called from list_notes
      //checking to see if variable note is a Note
      if(!variables.contains(note))
         Error.declaredOrInitialized(Error.NOT_DECLARED, note);
      else {//variable note is declared
         Object o = variables.returnVariable(note);
         if (o instanceof lilypond.Note){
            Note n = (lilypond.Note) o;
            if (n.initialized == true){
               Note addNote = n.cloneNote();
               Chord chord = new Chord(new Note[]{addNote});
               return chord;
            }
            else
               Error.declaredOrInitialized(Error.NOT_INITIALIZED, note);
         }
         else
            Error.isNot(note, "lilypond.Note");
      }
      return null;
   }
   /**
    * Creates new Chord containing a list_notes called from chord
    * @param var_name Chord variable name
```

```
     * @param list_notes List of notes
     */
    public void CHORD(String var_name, Chord list_notes)
    {// declaring and initializing a chord. called from chord
       if (variables.contains(var_name)){ //variable is declared
          Object o = variables.returnVariable(var_name);
          Error.alreadyDeclared(var_name, o.getClass().getName());
       }
       else if (list_notes != null){ //variable is not declared
          Chord chord = list_notes;
          chord.initialized = true;
          variables.addVariable(var_name, chord);
          //System.out.println(var_name + " == " + chord.printChord
() + "\n");
       }
    }
    /**
     * Creates a new null Chord, called from chord
     * @param var_name Chord variable name
     */
    public void CHORD(String var_name)
    {//declaring a chord without initializing it
       if (variables.contains(var_name)){ //variable is declared
          Object o = variables.returnVariable(var_name);
          Error.alreadyDeclared(var_name, o.getClass().getName());
       }
       else {//variable isn't already declared
          Chord chord = new Chord();
          chord.initialized = false;
          variables.addVariable(var_name, chord);
          System.err.println("Chord " + var_name + " declared but
not initialized.");
       }
    }
    /**
     * Re-initializes chord that was already declared, called from
chord
     * @param var_name Chord variable name
     * @param list_notes List of notes
     */
    public void INITIALIZE_CHORD(String var_name,  Chord list_notes)
```

```java
{

      if (variables.contains(var_name)){ //variable is declared
         Object o = variables.returnVariable(var_name);
         if(o instanceof lilypond.Chord){
            if(list_notes!=null){
               Chord chord = list_notes;
               chord.initialized = true;
               variables.addVariable(var_name, chord);
               //System.out.println(var_name + " == " + chord.printChord() + "\n");
            }
         }
         else
            Error.alreadyDeclared(var_name, o.getClass().getName());
      }
      else
         Error.declaredOrInitialized(Error.NOT_DECLARED, var_name);

   }
   /**
    *  Adds to a list of chords by concatenating the name to the current list, called from list_chords
    * @param name1 New name to add
    * @param name2 Current list of names
    * @return Returns concatenated list
    */
   public String MULT_CHORDS(int name1, String name2) {
      // TODO Auto-generated method stub
      return name2.concat(","+name1);
   }
   /**
    * Adds to a list of chords one chord, called from list_chords
    * @param name Chord name
    * @return list
    */
   public String ONE_CHORDS(int name) {
      // TODO Auto-generated method stub
      return Integer.toString(name);
   }
   /**
    * Creates a new sequence, called from sequence
    * @param var_name Sequence variable name
    */
   public void SEQUENCE(String var_name)
   {//declaring and initializing a sequence
      if (variables.contains(var_name)) {
         Object o = variables.returnVariable(var_name);
         Error.alreadyDeclared(var_name, o.getClass().getName());
      }
      else {
         Sequence sequence = new Sequence();
         variables.addVariable(var_name, sequence);
         sequence.initialized = true;
         //System.out.println(var_name +" == " +
sequence.printSequence() +"\n");
      }
   }
   /**
    * Creates a new null sequence, called from sequence
    * @param name Sequence variable name
    */
   public void SEQ(String name)
   {//declaring without initializing
      if (variables.contains(name)){
         Object o = variables.returnVariable(name);
         Error.alreadyDeclared(name, o.getClass().getName());
      }
      else {
         Sequence seq = new Sequence(1);
         seq.initialized = false;
         variables.addVariable(name,seq);
         System.err.println(seq + " has been declared but not
initialized.");
      }
   }
   /**
    * Adds to a sequence a list of variables (chords, notes,
sequences), called from add_to_sequence
    * @param sequence_name Sequence variable name
    * @param variableList Variable list
    */
   public void ADD_TO_SEQUENCE(String sequence_name, ArrayList<Object> variableList)
   {
      Object o = variables.returnVariable(sequence_name);
      if(variables.contains(sequence_name)){

         if(o instanceof lilypond.Sequence){
            Sequence sequence = (Sequence) variables.returnVariable(sequence_name) ;
            if(sequence.initialized){
               if(variableList!=null){
                  for(int i = variableList.size() - 1 ; i >= 0 ; i--)
                  {
                     Object object = variableList.get(i) ;

                     if(object instanceof lilypond.Note)
                     {
                        Note note = (Note) object ;
                        Note inputNote = note.cloneNote() ;

                        sequence.add(Chord.convertNote(inputNote)) ;
                     }
                     if(object instanceof lilypond.Chord)
                     {
                        Chord chord = (Chord) object ;
                        Chord inputChord = chord.cloneChord() ;

                        sequence.add(inputChord) ;
                     }
                     if(object instanceof lilypond.Sequence)
                     {
                        Sequence sequencetoAdd = (Sequence) object ;
                        sequence = Sequence.concatenateSequences(sequence, sequencetoAdd) ;
                        sequence.initialized=true;
                        variables.removeVariable(sequence_name) ;
                        variables.addVariable(sequence_name,
sequence) ;
                     }
                  }
               }
            }
            else
               Error.declaredOrInitialized(Error.NOT_INITIALIZED,
sequence_name);
            //System.out.println(sequence_name + " == " +
sequence.printSequence() + "\n") ;
         }
         else if(o==null){
            Error.declaredOrInitialized(Error.NOT_DECLARED,
sequence_name);
         }
         else{
            Error.mismatched("Add to sequence", o.getClass
().getName());
         }

      }
      else{
         Error.declaredOrInitialized(Error.NOT_DECLARED,
sequence_name);
      }

//*Richard*//

   }
   /**
    * Returns a chord in a specified sequence, called from
chord_to_sequence
    * @param name1 Sequence variable name
    * @param num Chord number in specified sequence
    * @return Returns specified chord
    */
   public Chord CHORD_IN_SEQ(String name1, int num) {
      // TODO Auto-generated method stub
      Object o = variables.returnVariable(name1);

      if(o instanceof lilypond.Sequence){
         Sequence s = (Sequence) o;
         if(s.initialized)
            return s.chordAt(num);
         else{
            Error.declaredOrInitialized(Error.NOT_INITIALIZED, name1);
            return null;
         }
      }
      else{
         if(!variables.contains(name1))
            Error.declaredOrInitialized(Error.NOT_DECLARED, name1);
         else
            Error.mismatched("Chord In Sequence", o.getClass().getName());
         return null;
      }
   }
   /**
    * Returns a note in a specified sequence, called from note_in_seq
    * @param name1 Sequence variable name
    * @param num_chord Chord number in specified sequence
    * @param num_note Note number in specified chord
    * @return Returns specified note
    */
   public Note NOTE_IN_SEQ(String name1, int num_chord, int num_note) {
      // TODO Auto-generated method stub
      Object o = variables.returnVariable(name1);
      if(o instanceof lilypond.Sequence){
         Sequence s = (Sequence) o;
         if(s.initialized)
            return s.chordAt(num_chord).noteAt(num_note);
         else{
            Error.declaredOrInitialized(Error.NOT_INITIALIZED, name1);
            return null;
         }
      }
      else
      {
         if(!variables.contains(name1))
            Error.declaredOrInitialized(Error.NOT_DECLARED, name1);
         else
            Error.mismatched("Note In Sequence",o.getClass
().getName());
         return null;
      }
   }
   /**
    * Subsequence function to return a subsequence of a specified
sequence with a start and end index, called from sub_seq
    * @param name Sequence variable name
    * @param num1 Start index
    * @param num2 End index
    * @return Returns a sequence subsequence
    */
   public Sequence SUB_SEQ(String name, int num1,int num2) {
      Object o = variables.returnVariable(name);
      if(variables.contains(name)){
         if(o instanceof lilypond.Sequence){
            Sequence s = (Sequence) variables.returnVariable(name);
            if(s.initialized){
               return s.subsequence(num1, num2);
            }
            else
            {
               Error.declaredOrInitialized(Error.NOT_INITIALIZED,
name);
               return null;
            }
         }
         else{
            Error.mismatched("Subsequence", o.getClass().getName
());
            return null;
         }
      }
      else{
         Error.declaredOrInitialized(Error.NOT_DECLARED, name);
         return null;
```

```
        }
    }
    /**
     * Subset function to return a sequence of a specified list of chords, called from sub_seq
     * @param name Sequence variable name
     * @param chordlist List of chords to add to the sequence
     * @return Returns a sequence subset
     */
    public Sequence SUB_SET(String name, String chordlist) {
        // TODO Auto-generated method stub
        Object o = variables.returnVariable(name);
        if(variables.contains(name)){
            if(o instanceof lilypond.Sequence){
                Sequence s = (Sequence) variables.returnVariable(name);
                if(s.initialized){
                    StringTokenizer toke = new StringTokenizer(chordlist,",");
                    int num = toke.countTokens();
                    int[]chordslist = new int[num];
                    for(int i = 0;i<num;i++){
                        chordslist[i] = Integer.parseInt(toke.nextToken());
                    }
                    return s.sequenceFrom(chordslist);
                }
                else{
                    Error.declaredOrInitialized(Error.NOT_INITIALIZED, name);
                    return null;
                }
            }
            else{
                Error.mismatched("Subset", o.getClass().getName());
                return null;
            }
        }
        else{
            Error.declaredOrInitialized(Error.NOT_DECLARED, name);
            return null;
        }
    }
    /**
     * Concatenation function of two specified sequences, called from concat_seq
     * @param s1 Sequence variable name 1
     * @param s2 Sequence variable name 2
     * @return Concatenation sequence of s1 and s2
     */
    public Sequence CONCAT_SEQ(String s1, String s2){

        Object o1 = variables.returnVariable(s1);
        Object o2 = variables.returnVariable(s2);
        if(variables.contains(s1) && variables.contains(s2)){

            if(o1 instanceof lilypond.Sequence && o2 instanceof lilypond.Sequence){
                Sequence newS, sequence1, sequence2;
                sequence1 = (Sequence) variables.returnVariable(s1);
                sequence2 = (Sequence) variables.returnVariable(s2);
                if(sequence1.initialized || sequence2.initialized){
                    newS = Sequence.concatenateSequences(sequence1, sequence2);

                    //System.out.println(newS.printSequence());


                    return newS ;
                }
                else
                {
                    if(!sequence1.initialized)
                        Error.declaredOrInitialized(Error.NOT_INITIALIZED, s1);
                    if(!sequence2.initialized)
                        Error.declaredOrInitialized(Error.NOT_INITIALIZED,s2);
                    return null;
                }
            }
            else{
                if(!(o1 instanceof lilypond.Sequence)){
```

```
                Error.mismatched("Concatenated Sequence",
o1.getClass().toString());
            }
            if(!(o2 instanceof lilypond.Sequence)){
                Error.mismatched("Concatenated Sequence",
o2.getClass().toString());
            }
            return null;
        }
    }
    else{
        if(!variables.contains(s1)){
            Error.declaredOrInitialized(Error.NOT_DECLARED, s1);
        }
        if(!variables.contains(s2)){
            Error.declaredOrInitialized(Error.NOT_DECLARED, s2);
        }
        return null;
    }


}
/**
 * Adds one variable to the ArrayList for adding to a sequence
in add_to_sequence, called from list_variables
 * @param name Variable name to be added to list
 * @return Returns the list
 */
public ArrayList<Object> ONE_VARIABLE(String name)
{

    ArrayList<Object> variableList = new ArrayList<Object>() ;

    Object object = variables.returnVariable(name) ;
    if(object!=null){
        variableList.add(object) ;
        return variableList ;
    }
    else{
        Error.declaredOrInitialized(Error.NOT_DECLARED, name);
        return null;
```

```
      }

   }
   /**
    * Adds a variable to an ArrayList that has already been created in add_to_sequence, called from list_variables
    * @param var_left Variable name to be added to list
    * @param variableList Current variable list
    * @return Returns the list
    */
   public ArrayList<Object> MULT_VARIABLES(String var_left, ArrayList<Object> variableList)
   {
      Object object = variables.returnVariable(var_left) ;
      if(object!=null){
         variableList.add(object) ;

         return variableList ;
      }
      else{
         Error.declaredOrInitialized(Error.NOT_DECLARED, var_left);
         return null;
      }
   }
   /**
    * Adds an object to an ArrayList for adding to a sequence in add_to_sequence, called from list_variables
    * @param object Variable object name to be added
    * @return Returns the list
    */
   public ArrayList<Object> ONE_VARIABLE_N(Object object)
   {
      ArrayList<Object> variableList = new ArrayList<Object>() ;

      variableList.add(object) ;
      return variableList ;


   }
   /**
    * Adds an object to an ArrayList that has already been created
```

```
in add_to_sequence, called from list_variables
     * @param object Variable object name to be added
     * @param variableList Current variable list
     * @return Returns the list
     */
    public ArrayList<Object> MULT_VARIABLE_N(Object object,
ArrayList<Object> variableList)
    {
       variableList.add(object) ;
       return variableList ;


//*Jonathan*//

    }
    /**
     * Changes the attribute color of a specified note, called from
assign
     * @param var_name Note variable name
     * @param attribute Color attribute name
     */
    public void ATTRIBUTE_COLOR(String var_name, String attribute)
    {
       if (variables.contains(var_name)){
          Object o = variables.returnVariable(var_name);
          if(o instanceof lilypond.Note){
             Note note = (lilypond.Note) o;
             if (note.initialized == true){
                note.Color = reference.checkColor(attribute,1);
                //System.out.println(note.printNote());
             }
             else
                Error.declaredOrInitialized(Error.NOT_INITIALIZED,
var_name);
          }
          else
             Error.mismatched("<", o.getClass().getName());
       }
       else
          Error.declaredOrInitialized(Error.NOT_DECLARED, var_name);
    }
    /**
     * Changes the attribute instrument of a specified note, called from assign
     * @param var_name Note variable name
     * @param attribute Instrument attribute name
     */
    public void ATTRIBUTE_INSTRUMENT(String var_name, String attribute)
    {
       if (variables.contains(var_name)){
          Object o = variables.returnVariable(var_name);
          if(o instanceof lilypond.Note){
             Note note = (lilypond.Note) o;
             if (note.initialized == true){
                note.Instrument = reference.checkInstrument(attribute,1);
                //System.out.println(note.printNote());
             }
             else
                Error.declaredOrInitialized(Error.NOT_INITIALIZED, var_name);
          }
          else
             Error.mismatched("<", o.getClass().getName());
       }
       else
          Error.declaredOrInitialized(Error.NOT_DECLARED, var_name);
    }
    /**
     * Changes the attribute shape of a specfied note, called from assign
     * @param var_name Note variable name
     * @param attribute Shape attribute name
     */
    public void ATTRIBUTE_SHAPE(String var_name, String attribute)
    {
       if (variables.contains(var_name)){
          Object o = variables.returnVariable(var_name);
          if(o instanceof lilypond.Note){
             Note note = (lilypond.Note) o;
             if (note.initialized == true){
                note.Shape = reference.checkShape(attribute,1);
                //System.out.println(note.printNote());
            }
            else
                Error.declaredOrInitialized(Error.NOT_INITIALIZED,
var_name);
        }
        else
            Error.mismatched("<", o.getClass().getName());
    }
    else
        Error.declaredOrInitialized(Error.NOT_DECLARED, var_name);
}
/**
 * Assigns a note to a note, called from var
 * @param name1 String Note variable name 1
 * @param name2 String Note variable name 2
 * @param create Boolean tag if name1 needs to be created
 */
public void ASSIGN_VAR_NOTE(String name1, String name2, boolean
create){
    Object o2 = variables.returnVariable(name2);
    Object o1 = variables.returnVariable(name1);
    if(o2 instanceof lilypond.Note){
        if(create){
            if(!variables.contains(name1)){
                Note v1 ;

                Note v2 =(Note) variables.returnVariable(name2);
                if(v2.initialized){
                    v1 = v2.cloneNote();
                    variables.addVariable(name1, v1);
                    //System.out.println(v1.printNote()+ "=="+
v2.printNote());
                }
                else
                    Error.declaredOrInitialized
(Error.NOT_INITIALIZED, name2);
            }
            else
                Error.alreadyDeclared(name1,o1.getClass().getName
());
        }
        else{
            if(o1 instanceof lilypond.Note){
                Note v1 ;
                Note v2 =(Note) variables.returnVariable(name2);
                if(v2.initialized){
                    v1 = v2.cloneNote();
                    variables.removeVariable(name1);
                    variables.addVariable(name1, v1);
                    //System.out.println(v1.printNote()+ "=="+ v2.printNote());
                }
                else
                    Error.declaredOrInitialized(Error.NOT_INITIALIZED, name2);
            }
            else
                Error.mismatched("Assign note", o1.getClass().getName());
        }
    }
    else
    {
        if(!variables.contains(name2)){
            Error.declaredOrInitialized(Error.NOT_DECLARED, name2);
        }
        else
            Error.mismatched("Assign note", o2.getClass().getName());
    }
}
/**
 * Assigns a note to a note, called from var
 * @param name1 String Note variable name 1
 * @param name2 Note variable name 2
 * @param create Boolean tag if name1 needs to be created
 */
public void ASSIGN_VAR_NOTE(String name1, Note name2, boolean create) {
    // TODO Auto-generated method stub
```

```java
      Object o1 = variables.returnVariable(name1);

      if(create){
         if(!variables.contains(name1)){
            if(name2.initialized){
               Note v1 = name2.cloneNote();
               variables.addVariable(name1, v1);
               //System.out.println(v1.printNote()+ "=="+ name2.printNote());
            }
            else
               Error.nullPointer();
         }
         else{
            Error.alreadyDeclared(name1,o1.getClass().getName());
         }
      }
      else{
         if(o1 instanceof lilypond.Note){
            Note v1 = name2.cloneNote();
            if(name2.initialized){
               variables.removeVariable(name1);
               variables.addVariable(name1, v1);
               //System.out.println(v1.printNote()+ "=="+ name2.printNote());
            }
            else
               Error.nullPointer();
         }
         else{
            Error.mismatched("Assign note", o1.getClass().getName());
         }
      }
   }
   /**
    * Assigns a chord to a chord, called from var
    * @param name1 String Chord variable name 1
    * @param name2 String Chord variable name 2
    * @param create Boolean tag if name1 needs to be created
    */
```

```java
  public void ASSIGN_VAR_CHORD(String name1, String name2, boolean create){
      Object o2 = variables.returnVariable(name2);
      Object o1 = variables.returnVariable(name1);
      if(o2 instanceof lilypond.Chord){
         if(create){
            if(!variables.contains(name1)){
               Chord v1;
               Chord v2 =(Chord) variables.returnVariable(name2);
               if(v2.initialized){
                  v1 = v2.cloneChord();
                  variables.addVariable(name1, v1);
                  //System.out.println(v1.printChord()+ "=="+ v2.printChord());
               }
               else
                  Error.declaredOrInitialized(Error.NOT_INITIALIZED, name2);
            }
            else
            {
               Error.alreadyDeclared(name1,o1.getClass().getName());
            }
         }
         else{
            if(o1 instanceof lilypond.Note){
               Chord v1;
               Chord v2 =(Chord) variables.returnVariable(name2);
               if(v2.initialized){
                  v1 = v2.cloneChord();
                  variables.removeVariable(name1);
                  variables.addVariable(name1, v1);
                  //System.out.println(v1.printChord()+ "=="+ v2.printChord());
               }
               else{
                  Error.declaredOrInitialized(Error.NOT_INITIALIZED, name2);
               }
            }
```

```
                else{
                    Error.mismatched("Assign chord", o1.getClass
().getName());
                }

            }
        }
        else
        {
            if(!variables.contains(name2)){
                Error.declaredOrInitialized(Error.NOT_DECLARED, name2);
            }
            else{
                Error.mismatched("Assign chord", o2.getClass().getName
());
            }
        }
    }
    /**
     * Assigns a chord to a chord, called from var
     * @param name1 String Chord variable name 1
     * @param name2 Chord variable name 2
     * @param create Boolean tag if name1 needs to be created
     */
    public void ASSIGN_VAR_CHORD(String name1, Chord name2, boolean
create) {
        // TODO Auto-generated method stub
        Object o1 = variables.returnVariable(name1);
        if(create){
            if(!variables.contains(name1)){
                Chord v1;
                if(name2.initialized){
                    v1 = name2.cloneChord();
                    variables.addVariable(name1, v1);
                    //System.out.println(v1.printChord()+ "=="+
name2.printChord());
                }
                else
                    Error.nullPointer();
            }
            else{
                Error.alreadyDeclared(name1,o1.getClass().getName());
            }
        }
        else{
            if(o1 instanceof lilypond.Chord){
                Chord v1;
                if(name2.initialized){
                    v1 = name2.cloneChord();
                    variables.removeVariable(name1);
                    variables.addVariable(name1, v1);
                    //System.out.println(v1.printChord()+ "=="+
name2.printChord());
                }
                else{
                    Error.nullPointer();
                }
            }
            else
                Error.mismatched("Assign chord", o1.getClass().getName
());
        }
    }

    //*Taylor*//

    /**
     * Assigns a sequence to a sequence, called from var
     * @param name1 String Sequence variable name 1
     * @param name2 String Sequence variable name 2
     * @param create Boolean tag if name1 needs to be created
     */
    public void ASSIGN_VAR_SEQ(String name1, String name2, boolean
create){
        Object o2 = variables.returnVariable(name2);
        Object o1 = variables.returnVariable(name1);
        if(o2 instanceof lilypond.Sequence){
            if(create){
                if(!variables.contains(name1)){
                    Sequence v1;

                    Sequence v2 =(Sequence) variables.returnVariable
(name2);
                    if(v2.initialized){
                        v1 = v2.cloneSequence();
                        variables.addVariable(name1, v1);
                        //System.out.println(v1.printSequence()+ "=="+
v2.printSequence());
                    }
                    else
                        Error.declaredOrInitialized
(Error.NOT_INITIALIZED, name2);
                }
                else
                    Error.alreadyDeclared(name1,o1.getClass().getName
());
            }
            else{
                if(o1 instanceof lilypond.Sequence){
                    Sequence v1;
                    Sequence v2 =(Sequence) variables.returnVariable
(name2);
                    if(v2.initialized){
                        v1 = v2.cloneSequence();
                        variables.addVariable(name1, v1);
                        //System.out.println(v1.printSequence()+ "=="+
v2.printSequence());
                    }
                    else
                        Error.declaredOrInitialized
(Error.NOT_INITIALIZED, name2);
                }
                else
                    Error.mismatched("Assign sequence", o1.getClass
().getName());
            }
        }
        else{
            if(!variables.contains(name2)){
                Error.declaredOrInitialized(Error.NOT_DECLARED, name2);
            }
            else
                Error.mismatched("Assign sequence", o2.getClass
```

```java
().getName());
        }
    }
    /**
     * Assigns a sequence to a sequence, called from var
     * @param name1 String Sequence variable name 1
     * @param name2 Sequence variable name 2
     * @param create Boolean tag if name1 needs to be created
     */
    public void ASSIGN_VAR_SEQ(String name1, Sequence name2, boolean create) {
        // TODO Auto-generated method stub
        Object o1 = variables.returnVariable(name1);
        if(create){
            if(!variables.contains(name1)){
                if(name2.initialized){
                    Sequence v1 = name2.cloneSequence();

                    variables.addVariable(name1, v1);
                    //System.out.println(v1.printSequence()+ "=="+ name2.printSequence());
                }
                else
                    Error.nullPointer();
            }
            else
                Error.alreadyDeclared(name1,o1.getClass().getName());
        }
        else{
            if(o1 instanceof lilypond.Sequence){
                if(name2.initialized){
                    Sequence v1 = name2.cloneSequence();
                    variables.removeVariable(name1);
                    variables.addVariable(name1, v1);
                    //System.out.println(v1.printSequence()+ "=="+ name2.printSequence());
                }
                else
                    Error.nullPointer();
            }
            else
            Error.mismatched("Assign sequence", o1.getClass().getName());
        }
    }
    /**
     * Assigns a variable to a variable, checks to see which type is being assigned, called from var
     * @param name1 String var name 1
     * @param name2 String var name 2
     */
    public void ASSIGN_VAR(String name1, String name2){ // ADD error of "a=ASSIGN"

        //check to see if they are of the same type
        Object o1 = variables.returnVariable(name1);
        Object o2 = variables.returnVariable(name2);
        if(variables.contains(name1)&&variables.contains(name2)){
            if(o1 instanceof lilypond.Note && o2 instanceof lilypond.Note)
            {
                Note v2 =(Note) o2 ;
                if(v2.initialized){
                    Note v1 = v2.cloneNote();

                    variables.removeVariable(name1) ; //add error here
                    variables.addVariable(name1, v1) ;
                    //System.out.println(v1.printNote()+ "=="+ v2.printNote());
                }
                else
                    Error.declaredOrInitialized(Error.NOT_INITIALIZED, name2);
            }
            else if(o1 instanceof lilypond.Chord&& o2 instanceof lilypond.Chord)
            {
                Chord v2 =(Chord) o2 ;
                if(v2.initialized){
                    Chord v1 = v2.cloneChord();

                    variables.removeVariable(name1) ;
                    variables.addVariable(name1, v1) ;
                    //System.out.println(v1.printChord()+ "=="+
    v2.printChord());
                }
                else
                    Error.declaredOrInitialized(Error.NOT_INITIALIZED,
    name2);
            }
            else if(o1 instanceof lilypond.Sequence&& o2 instanceof
    lilypond.Sequence)
            {
                Sequence v2 =(Sequence) o2 ;
                if(v2.initialized){
                    Sequence v1 = v2.cloneSequence();

                    variables.removeVariable(name1) ;
                    variables.addVariable(name1, v1) ;
                    //System.out.println(v1.printSequence()+ "=="+
    v2.printSequence());
                }
                else
                    Error.declaredOrInitialized(Error.NOT_INITIALIZED,
    name2);
            }
            else if(o1 instanceof Integer) ///ERROR CHECKING!
            {
                int v1 = (Integer) o1;

                variables.removeVariable(name1);
                variables.addVariable(name1, v1);
            }
            else if(o1 instanceof Boolean)
            {
                boolean v1 = (Boolean) o1;

                variables.removeVariable(name1);
                variables.addVariable(name1, v1);
            }
            else{
                Error.sameType(name1, name2);
            }
```

```java

        }
        else
        {
           if(!variables.contains(name1)){
              Error.declaredOrInitialized(Error.NOT_DECLARED, name1);
              if(!variables.contains(name2)){
                 Error.declaredOrInitialized(Error.NOT_DECLARED, 
name2);
              }
           }
           else if(!variables.contains(name2)){
              if(!variables.contains(name1))
                 Error.declaredOrInitialized(Error.NOT_DECLARED, 
name1);
              Error.declaredOrInitialized(Error.NOT_DECLARED, name2);
           }
           else{
              Error.sameType(name1, name2);
           }
        }


     }

     //*Farbound*//

     /**
      * Foreach function to change all the shapes in a specified 
sequence, called from foreach
      * @param name Sequence variable name
      * @param attribute Shape attribute name
      */
     public void FOREACH_SHAPE(String name, String attribute) {
        // TODO Auto-generated method stub
        Object o = variables.returnVariable(name);
        if(o instanceof lilypond.Sequence){
           Sequence s = (Sequence) o;
           if(s.initialized){
              Iterator<Chord> it = s.s.iterator();
              while(it.hasNext()){
                    Chord temp = it.next();
                    for(int j = 0; j< temp.Notes.length;j++){
                        temp.Notes[j].Shape = reference.checkShape(attribute, 1);
                    }
                }
            }
            else
                Error.declaredOrInitialized(Error.NOT_INITIALIZED, name);
        }

        else{
            Error.mismatched("foreach", o.getClass().getName());
        }
    }
    /**
     * Foreach function to change all the colors in a specified sequence, called from foreach
     * @param name Sequence variable name
     * @param attribute Color attribute name
     */
    public void FOREACH_COLOR(String name, String attribute) {
        // TODO Auto-generated method stub
        Object o = variables.returnVariable(name);
        if(o instanceof lilypond.Sequence){
            Sequence s = (Sequence) o;
            if(s.initialized){
                Iterator<Chord> it = s.s.iterator();
                while(it.hasNext()){
                    Chord temp = it.next();
                    for(int j = 0; j< temp.Notes.length;j++){
                        temp.Notes[j].Color = reference.checkColor(attribute, 1);
                    }
                }
            }
            else
                Error.declaredOrInitialized(Error.NOT_INITIALIZED, name);
        }
```

```

      else{
         Error.mismatched("foreach", o.getClass().getName());
      }
   }
   /**
    * Foreach function to change all the instruments in a specified sequence, called from foreach
    * @param name Sequence variable name
    * @param attribute Instrument attribute name
    */
   public void FOREACH_INSTRUMENT(String name, String attribute) {
      // TODO Auto-generated method stub
      Object o = variables.returnVariable(name);
      if(o instanceof lilypond.Sequence){
         Sequence s = (Sequence) o;
         if(s.initialized){
            Iterator<Chord> it = s.s.iterator();
            while(it.hasNext()){
               Chord temp = it.next();
               for(int j = 0; j< temp.Notes.length;j++){
                  temp.Notes[j].Instrument = reference.checkInstrument(attribute, 1);
               }
            }
         }
         else
            Error.declaredOrInitialized(Error.NOT_INITIALIZED, name);
      }

      else{
         Error.mismatched("foreach", o.getClass().getName());
      }
   }
   /**
    * Changes the duration of a specified note, called from change_duration
    * @param name Note variable name
    * @param num Duration change integer
    */
```

```java
    public void CHANGE_DURATION(String name, int num)
    {
        if(!variables.contains(name)) //variable name is not declared
            Error.declaredOrInitialized(Error.NOT_DECLARED, name);
        else{ //variable is declared
            Object o = variables.returnVariable(name);
            if (o instanceof lilypond.Note){ //checks if variable is a note
                Note note = (lilypond.Note) o;
                if (note.initialized == false) //variable has not been initialized
                    Error.declaredOrInitialized(Error.NOT_INITIALIZED, name);
                else{ //variable has been initialized
                    note.Duration = num;
                }
            }
            else //variable is not of type lilypond.Note
                Error.mismatched("^", o.getClass().getName());
        }
    }
    /**
     * Displays the specified sequence, called from display
     * @param var_name Sequence variable name
     * @param header Header string to be displayed
     * @throws IOException
     */
    public void DISPLAY(String var_name, String header) throws IOException
    {
        Object o = variables.returnVariable(var_name);

        if(o instanceof lilypond.Sequence && header instanceof String){
            Sequence outputS = (Sequence) o ;
            if(outputS.initialized){
                LilypondConvert.writeFiles(outputS,header) ;

                LilypondConvert.executeLilypond() ;
            }
            else
            Error.declaredOrInitialized(Error.NOT_INITIALIZED, var_name);
        }
        else
        {
            if(o==null)//checks if variable is null
                Error.declaredOrInitialized(Error.NOT_DECLARED, var_name);
            else
                Error.mismatched("display", o.getClass().getName());
        }
    }

    public static class Variable
    {
        LinkedHashMap<String, Object> variables ;

        /**
         * Constructor for Variable table, creates LinkedHashMap of variables
         */
        public Variable()
        {
            variables = new LinkedHashMap<String, Object>() ;
        }
        /**
         * Adds a variable to the variable table
         * @param name Variable name
         * @param object Object type
         */
        public void addVariable(String name, Object object)
        {
            variables.put(name, object) ;
        }
        /**
         * Removes specified variable from the variable table
         * @param name Variable name
         */
        public void removeVariable(String name)
        {
            variables.remove(name) ;
        }
        /**
         * Returns specified variable from variable table
         * @param name Variable name
         * @return specified variable
         */
        public Object returnVariable(String name)
        {
           Object variable = variables.get(name) ;
           return variable ;
        }
        /**
         * Boolean function that returns true if the specified variable is in the the variable list
         * @param name Variable name
         * @return true/false
         */
        public boolean contains(String name){
           return variables.containsKey(name);
        }
     }


   }

```

```
//*Jonathan*//

package compiler;

public class Error
{
    /**
     * Type mismatch output for an operator
     *
     * @param op operator that is mismatch
     * @param type
     */
    public static void mismatched(String op, String type)
    {
        System.err.println("Error: The " + op
            + " operator isn't defined for type " + type + ".");
    }

    /**
     * Output for errors where a variable is not declared or
initialized
     *
     * @param code Error code
     * @param vName Variable name
     */
    public static void declaredOrInitialized(int code, String vName)
    {
        System.err.println("Error: Variable " + vName + errorMessages
[code]);
    }

    /**
     * Error for output when variables are not of the same type
     *
     * @param vName1 Variable1
     * @param vName2 Variable2
     */
    public static void sameType(String vName1, String vName2)
    {
        System.err.println("Error: Variable " + vName1 + " and " +
vName2
```

```
            + " are not of the same type.");
    }

    /**
     * Error for output when a variable has already been declared
     *
     * @param name Variable name
     * @param type Variable type
     */
    public static void alreadyDeclared(String name, String type)
    {
        System.err.println("Error: " + type + " " + name
            + " has already been declared.");
    }

    /**
     * Error output for null point
     */
    public static void nullPointer()
    {
        System.err.println("Error: Null Pointer Exception");
    }

    /**
     * Error output for variable being of wrong type
     *
     * @param name Variable name
     * @param type Specified type
     */
    public static void isNot(String name, String type)
    {
        System.err.println("Error: " + name + " is not of type " + type
+ ".");
    }

    public static final int NOT_DECLARED = 0, NOT_INITIALIZED = 1;
    private static final String[] errorMessages = { " has not been
declared.",
            " has not been initialized." };
}

```

```java
//*Taylor*//

package compiler;

import java.io.*;

public class Main
{
    public static void main(String[] args) throws java.lang.Exception
    {
        Symbol sym;
        try
        {
            /*// Uncomment to check the tokens the lexical analyzer produces
            Yylex lexer = new Yylex(new FileReader("test.txt"));
            for (sym = lexer.next_token(); sym.sym != 0; sym = lexer.next_token())
            {
                System.out.println("Token " + sym + ", with value = " + sym.value
                    + "; at line " + sym.left + ", column " + sym.right);
            }
            System.out.println("Lexer has finished\n\n");
            */

            // open input file
            FileReader inFile = null;
            try
            {
                inFile = new FileReader("src/testfiles/Twinkle.mus");
            }
            catch (FileNotFoundException ex)
            {
                System.err.println("File " + "test" + " not found.");
                System.exit(-1);
            }


            Yylex lexer2;

        try
        {
           lexer2 = new Yylex(inFile);
           ParserCup p = new ParserCup(lexer2);
           p.parse();
        }
        catch (Exception lex)
        {
           // lex.printStackTrace() ;
           System.err.println("Error in your code");
        }
     }
     catch (Exception e)
     {
        // e.printStackTrace() ;
        System.err.println("Error in your code");
     }
  }
}
```

Front-End
Files generated automatically

```java

//----------------------------------------------------
// The following code was generated by CUP v0.11a beta 20060608
// Sat May 07 21:28:13 EDT 2011
//----------------------------------------------------

package compiler;

/** CUP generated interface containing symbol constants. */
public interface ParserSym {
  /* terminals */
  public static final int RBRACK = 20;
  public static final int SEQUENCE = 11;
  public static final int SEMICOLON = 13;
  public static final int PLUS = 15;
  public static final int RPAREN = 18;
  public static final int DIGIT = 16;
  public static final int INSTRUMENT = 8;
  public static final int CHORD = 10;
  public static final int LBRACK = 19;
  public static final int COLOR = 6;
  public static final int LPAREN = 17;
  public static final int COLON = 14;
  public static final int ID = 3;
  public static final int NOTE = 9;
  public static final int COMMA = 12;
  public static final int EOF = 0;
  public static final int FOREACH = 25;
  public static final int GTHAN = 22;
  public static final int CHANGE_DURATION = 24;
  public static final int SHAPE = 7;
  public static final int LTHAN = 21;
  public static final int error = 1;
  public static final int ADD_TO_SEQUENCE = 23;
  public static final int ASSIGN = 5;
  public static final int DISPLAY = 26;
  public static final int NEW = 4;
  public static final int STRING_TEXT = 2;
}


```

Back-End

```java
/**
 * Class that represents a note
 */
//*Irene*//
package lilypond;

public class Note
{
   public String Color, Shape, Pitch, Instrument;
   public int Duration;
   public int Octave;
   public boolean initialized;

   /**
    * A note object constructor Also adds the correct syntax for accidentals to
    * later be outputed in lilypond 'is' represents a sharp, 'isis' two sharps,
    * 'es' a flat, 'eses' two flats
    *
    * @param color the color assigned to the note
    * @param shape the shape assigned to the note
    * @param pitch the pitch of the note
    * @param duration the duration of the note
    * @param instrument the instrument assigned to the note
    * @param octave the octave of the note
    */
   public Note(String color, String shape, String pitch, int duration,
      String instrument, int octave)
   {
      // color
      Color = color;
      // shape
      Shape = shape;
      // pitch
      Pitch = pitch.toLowerCase();
      if (correctFormat() != true)
      {
         String newPitch = "";
         if (pitch.endsWith("#"))
```

```
            {
               newPitch = pitch.subSequence(0, 1) + "is";
            }
            if (pitch.endsWith("##"))
            {
               newPitch = pitch.subSequence(0, 1) + "isis";
            }
            if (pitch.endsWith("b"))
            {
               newPitch = pitch.subSequence(0, 1) + "es";
            }
            if (pitch.endsWith("bb"))
            {
               newPitch = pitch.subSequence(0, 1) + "es";
            }
            Pitch = newPitch.toLowerCase();
         }
         else
         {
            Pitch = pitch.toLowerCase();
         }
         // duration
         Duration = duration;
         // instrument
         Instrument = instrument;
         Octave = octave;
      }

      /**
       * Dummy note constructor to check for errors
       */
      public Note()
      {
      }

      /**
       * Function to return duration of the note
       *
       * @return the duration of the note
       */
      public int length()
```

```
    {
        return Duration;
    }

    /**
     * Checks if the pitch of the note is represented with a string in the
     * correct format
     *
     * @return true: pitch in correct format; false: pitch in wrong format
     */
    public boolean correctFormat()
    {
        if (Pitch.endsWith("is") || Pitch.endsWith("isis")
            || Pitch.endsWith("es") || Pitch.endsWith("eses")
            || Pitch.length() == 1)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    /**
     * Clones a note object. Meaning it creates a new object with the same
     * attributes as 'this'
     *
     * @return a new note object
     */
    public Note cloneNote()
    {
        String cColor = this.Color;
        String cShape = this.Shape;
        String cPitch = this.Pitch;
        int cDuration = this.Duration;
        String cInstrument = this.Instrument;
        int cOctave = this.Octave;
```

```java
        Note clone = new Note(cColor, cShape, cPitch, cDuration,
cInstrument,
            cOctave);
        return clone;
    }

    /**
     * Prints the note, used in error checking. Prints out the memory
address of
     * the note object
     *
     * @return a string representing all the information in the note
     */
    public String printNote()
    {
        String note = "";
        note = Pitch + Octave + "-" + Duration + " i:" + Instrument +
" c:"
            + Color + " :s" + Shape;
        note = note + "\t[ " + this + " ]";
        return note;
    }
}
```

```java
/**
 * Class that represents a chord
 */
//*Richard*//
package lilypond;

public class Chord
{
   public Note[] Notes;
   public boolean initialized;

   /**
    * Constructor for chord
    *
    * @param notes array of notes
    */
   public Chord(Note[] notes)
   {
      Notes = notes;
   }

   /**
    * Empty constructor for chord
    */
   public Chord()
   {
   }

   /**
    * Adds a note to the chord
    *
    * @param newNote note being added
    */
   public void addNote(Note newNote)
   {
      Note[] newNotes = new Note[Notes.length + 1];
      newNotes[0] = newNote;
      for (int i = 1; i < Notes.length + 1; i++)
      {
         newNotes[i] = Notes[i - 1];
      }
      Notes = newNotes;
   }

   /**
    * Writes the lilypond code for each chord
    *
    * @return returning a string with lilypond code
    */
   public String write()
   {
      String returnMe = "\n\\set Staff.midiInstrument = #\""
         + Notes[0].Instrument + "\"\n" + "\\override NoteHead #'color = #"
         + Notes[0].Color + "\n" + "\\override NoteHead #'style = #'"
         + Notes[0].Shape + "\n" + "<";
      for (int i = 0; i < Notes.length; i++)
      {
         returnMe = returnMe.concat(Notes[i].Pitch
            + Reference.octaveMap.get(Notes[i].Octave) + " ");
      }
      returnMe = returnMe.concat(">" + Notes[0].Duration + "\n");
      return returnMe;
   }

   /**
    * Chord function that converts a note to a chord
    *
    * @param note input a Note
    * @return returns a Chord
    */
   static public Chord convertNote(Note note)
   {
      Note[] oneNote = { note };
      Chord chord = new Chord(oneNote);
      return chord;
   }

   /**
    * noteAt will return a note at a specified location in the chord
    *
```

```java
     * @param loc Note location
     * @return returns a Note at specified location
     */
    public Note noteAt(int loc)
    {
       try
       {
          return Notes[loc];
       }
       catch (ArrayIndexOutOfBoundsException e)
       {
          System.err.println("Warning: Note " + loc
             + " not found in specified chord.");
       }
       return null;
    }

    /**
     * Size of chord function, number of notes
     *
     * @return integer size
     */
    public int returnSize()
    {
       return Notes.length;
    }

    /**
     * Prints the chord as standard output to check the memory
 location
     *
     * @return returns a String of the chord data
     */
    public String printChord()
    {
       String chord = "{ ";
       for (int i = 0; i < Notes.length; i++)
       {
          chord = chord + "\n\t\t" + Notes[i].printNote();
       }
       chord = chord + "\n\t}\t[ " + this + " ]";
```

```
        return chord;
    }

    /**
     * Clones data in chord
     *
     * @return cloned chord
     */
    public Chord cloneChord()
    {
        Note[] cloneNotes = new Note[Notes.length];
        for (int i = 0; i < Notes.length; i++)
        {
            cloneNotes[i] = Notes[i].cloneNote();
        }
        Chord cloneChord = new Chord(cloneNotes);
        return cloneChord;
    }
}

```

```
 * Class that represents a sequence of notes and chords
package lilypond;

import java.util.ArrayList;

public class Sequence
{
    public ArrayList<Chord> s;
    public boolean initialized;
    int count = 0;

    /**
     * Constructor for Sequence, creates new ArrayList of Chords
     */
    public Sequence()
    {
        s = new ArrayList<Chord>();
    }

    //*Jonathan*//

    /**
     * Constructor to create a null Sequence
     *
     * @param i tag i
     */
    public Sequence(int i)
    {
    }

    //*Richard*//


    /**
     * Adds a new Chord to the Sequence
     *
     * @param newChord new Chord to be added
     */
    public void add(Chord newChord)
    {
        s.add(newChord);
```

```java
}

/**
 * Gives a chord at a specified location.
 *
 * @param loc integer location
 * @return the Chord at the location
 */
public Chord chordAt(int loc)
{
   try
   {
      return s.get(loc);
   }
   catch (IndexOutOfBoundsException e)
   {
      System.err.println("Warning: Chord " + loc
         + " not found in specified sequence.");
   }
   return null;
}

/**
 * Function to return the size of the sequence, number of chords.
 *
 * @return integer size
 */
public int returnSize()
{
   return s.size();
}

/**
 * Function to return the note within a specified chord
 *
 * @param chordLoc Chord location in sequence
 * @param noteLoc Note location in specified sequence
 * @return specified Note
 */
public Note noteAt(int chordLoc, int noteLoc)
{
   return chordAt(chordLoc).noteAt(noteLoc);
}

/**
 * Adds to sequence specified chords to be returned
 *
 * @param chords integer array of chords to be returned
 * @return returns new sequence
 */
public Sequence sequenceFrom(int[] chords)
{
   Sequence seq = new Sequence();
   for (int i = chords.length - 1; i >= 0; i--)
   {
      seq.add(this.chordAt(chords[i]));
   }
   return seq;
}

/**
 * Creates a subsequence from two specified start and end indices.
 *
 * @param index1 start index
 * @param index2 end index
 * @return returns a Sequence subsequence
 */
public Sequence subsequence(int index1, int index2) // subsequence(5,0)
{
   Sequence subsequence = new Sequence();
   for (int i = index1; i <= index2; i++)
   {
      if (this.chordAt(i) != null)
      {
         subsequence.add(this.chordAt(i));
      }
   }
   return subsequence;
}

```

```java
//*Irene*//

/**
 * Creates a subsequence from a specified starting location
 *
 * @param index1 start location
 * @return returns a Sequence subsequence
 */
public Sequence subsequence(int index1) // subsequence(5)
{
   if (index1 >= this.returnSize())
   {
      System.err.println("Warning: Index is out of bounds");
   }
   Sequence subsequence = new Sequence();
   for (int i = index1; i < this.returnSize(); i++)
   {
      if (this.chordAt(i) != null)
      {
         subsequence.add(this.chordAt(i));
      }
   }
   return subsequence;
}

/**
 * Prints to standard out to check for memory locations of sequence
 *
 * @return string of sequence data
 */
public String printSequence()
{
   String sequence = "";
   try
   {
      for (int i = 0; i < this.returnSize(); i++)
      {
         sequence = sequence + this.chordAt(i).printChord() + " ";
      }
   }
   catch (NullPointerException e)
   {
      System.err.println("Warning: Index is out of bounds");
   }
   sequence = sequence + "\t[ " + this + " ]";
   return sequence;
}

/**
 * Function to concatenate two specified sequence and return that sequence
 *
 * @param sequence1 Sequence 1
 * @param sequence2 Sequence 2
 * @return new Sequence containing sequence1 and sequence2, in that order.
 */
static public Sequence concatenateSequences(Sequence sequence1,
   Sequence sequence2)
{
   Sequence concatenation = new Sequence();
   for (int i = 0; i < sequence1.returnSize(); i++)
   {
      concatenation.add(sequence1.chordAt(i).cloneChord());
   }
   for (int i = 0; i < sequence2.returnSize(); i++)
   {
      concatenation.add(sequence2.chordAt(i).cloneChord());
   }
   return concatenation;
}

/**
 * Clones data in sequence
 *
 * @return cloned data
 */
public Sequence cloneSequence()
{
   Sequence cloneSequence = new Sequence();
```

```java
        for (int i = 0; i < this.returnSize(); i++)
        {
            cloneSequence.add(this.chordAt(i).cloneChord());
        }
        return cloneSequence;
    }
}

```

```java
/**
 * Class to store information about available note shapes, colors, octaves, pitch, and durations
 */

//*Irene*//
package lilypond;

import java.util.HashMap;

public class Reference
{
   /**
    * A list of all note shape types
    */
   public static String[] shapeTable = { "default", "altdefault", "baroque",
         "neomensural", "mensural", "petrucci", "harmonic", "harmonic-black",
         "harmonic-mixed", "diamond", "cross", "xcircle", "triangle", "slash" };

   /**
    * A list of all instruments in midi
    */
   public static String[] instrumentTable = { "acoustic grand", "contrabass",
         "lead 7 (fifths)", "bright acoustic", "tremolo strings",
         "lead 8 (bass+lead)", "electric grand", "pizzicato strings",
         "pad 1 (new age)", "honky-tonk", "orchestral harp", "pad 2 (warm)",
         "electric piano 1", "timpani", "pad 3 (polysynth)",
         "electric piano 2", "string ensemble 1", "pad 4 (choir)",
         "harpsichord", "string ensemble 2", "pad 5 (bowed)", "clav",
         "synthstrings 1", "pad 6 (metallic)", "celesta", "synthstrings 2",
         "pad 7 (halo)", "glockenspiel", "choir aahs", "pad 8 (sweep)",
         "music box", "voice oohs", "fx 1 (rain)", "vibraphone",
"synth voice",
            "fx 2 (soundtrack)", "marimba", "orchestra hit", "fx 3 
(crystal)",
            "xylophone", "trumpet", "fx 4 (atmosphere)", "tubular 
bells",
            "trombone", "fx 5 (brightness)", "dulcimer", "tuba", "fx 6 
(goblins)",
            "drawbar organ", "muted trumpet", "fx 7 (echoes)", 
"percussive organ",
            "french horn", "fx 8 (sci-fi)", "rock organ", "brass 
section",
            "sitar", "church organ", "synthbrass 1", "banjo", "reed 
organ",
            "synthbrass 2", "shamisen", "accordion", "soprano sax", 
"koto",
            "harmonica", "alto sax", "kalimba", "concertina", "tenor 
sax",
            "bagpipe", "acoustic guitar (nylon)", "baritone sax", 
"fiddle",
            "acoustic guitar (steel)", "oboe", "shanai", "electric 
guitar (jazz)",
            "english horn", "tinkle bell", "electric guitar (clean)", 
"bassoon",
            "agogo", "electric guitar (muted)", "clarinet", "steel 
drums",
            "overdriven guitar", "piccolo", "woodblock", "distorted 
guitar",
            "flute", "taiko drum", "guitar harmonics", "recorder", 
"melodic tom",
            "acoustic bass", "pan flute", "synth drum", "electric bass 
(finger)",
            "blown bottle", "reverse cymbal", "electric bass (pick)",
            "shakuhachi", "guitar fret noise", "fretless bass", 
"whistle",
            "breath noise", "slap bass 1", "ocarina", "seashore", "slap 
bass 2",
            "lead 1 (square)", "bird tweet", "synth bass 1", "lead 2 
(sawtooth)",
            "telephone ring", "synth bass 2", "lead 3 (calliope)", 
"helicopter",
            "violin", "lead 4 (chiff)", "applause", "viola", "lead 5
```

```
(charang)",
            "gunshot", "cello", "lead 6 (voice)" };

    /**
     * A list of all colors available to color notes
     */
    public static String[] colorTable = { "black", "darkyellow",
            "green", "red", "white", "yellow", "darkred", "darkgreen",
            "grey", "cyan", "blue", "darkblue", "darkmagenta", "darkcyan",
            "magenta" };

    /**
     * A list of all available note durations
     */
    public static String[] durationTable = { "1", "2", "4", "8", "16", "32", "64", "128" };

    /**
     * A list of all the octaves that can be represented
     */
    public static int[] octaveTable = { -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

    /**
     * A list of all available note pitches
     */
    public static String[] noteTable = { "a", "b", "c", "d", "e", "f", "g" };

    /**
     * A list of all note accidentals
     */
    public static String[] noteAccidentals = { "is", "es", "isis", "eses" };


    /**
     * Represents a Hash Map that will store the mapping of an octave in correct
     * lilypond syntax
```

```java
     */
    public static Map<Integer, String> octaveMap = new HashMap<Integer, String>();

    /**
     * Constructor for class. Available to make for easy access to storage tables
     */
    public Reference()
    {
        populateOctaveMap();
    }

    /**
     * Functions to add keys and values to the Octave Hash Map Maps a pitch
     * octave to correct lilypond syntax
     */
    public void populateOctaveMap()
    {
        octaveMap.put(-1, ",,,,");
        octaveMap.put(0, ",,,");
        octaveMap.put(1, ",,");
        octaveMap.put(2, ",");
        octaveMap.put(3, "");
        octaveMap.put(4, "'");
        octaveMap.put(5, "''");
        octaveMap.put(6, "'''");
        octaveMap.put(7, "''''");
        octaveMap.put(8, "'''''");
        octaveMap.put(9, "''''''");
    }

    /**
     * Function to check a correct shape has been chosen
     *
     * @param shape shape string to check
     * @param tag 1: represents a note object is initialized for the first time.
     *            0: represents its an additional change to a note object
    * @return the shape if it is correct or a default
    */
   public String checkShape(String shape, int tag)
   {
      for (int i = 0; i < shapeTable.length; i++)
      {
         if (shape.equals(shapeTable[i]))
         {
            return shape;
         }
      }
      if (tag == 1)
      {
         System.err.println("Warning: shape " + shape
            + " is not a valid shape. System has defaulted to 'default' shape");
      }
      return shapeTable[0];
   }

   /**
    * Function to check a correct instrument has been chosen
    *
    * @param instrument instrument string to check
    * @param tag 1: represents a note object is initialized for the first time.
    *           0: represents its an additional change to a note object
    * @return the instrument if it is correct or a default
    */
   public String checkInstrument(String instrument, int tag)
   {
      for (int i = 0; i < instrumentTable.length; i++)
      {
         if (instrument.equals(instrumentTable[i]))
         {
            return instrument;
         }
      }
      if (tag == 1)
      {
         System.err
            .println("Warning: instrument "
               + instrument
               + " is not a valid instrument. System has defaulted to 'acoustic grand' instrument");
      }
      return instrumentTable[0];
   }

   /**
    * Function to check a correct color has been chosen
    *
    * @param color color string to check
    * @param tag 1: represents a note object is initialized for the first time.
    *            0: represents its an additional change to a note object
    * @return the color if it is correct or a default
    */
   public String checkColor(String color, int tag)
   {
      for (int i = 0; i < colorTable.length; i++)
      {
         if (color.equals(colorTable[i]))
         {
            return color;
         }
      }
      if (tag == 1)
      {
         System.err.println("Warning: color " + color
            + " is not a valid color. System has defaulted to 'black' color");
      }
      return colorTable[0];
   }

   /**
    * Function to check a correct octave has been chosen
    *
    * @param octave octave to check
```

```
     * @return the octave if it is correct or a default
     */
    public int checkOctave(int octave)
    {
        for (int i = 0; i < octaveTable.length; i++)
        {
            if (octave == octaveTable[i])
            {
                return octave;
            }
        }
        System.err.println("Warning: octave " + octave
            + " is not a valid octave. System has defaulted to '4' octave");
        return 4;
    }
}
```

```
 * Class to write output into correct lilypond format
package lilypond;

import java.io.*;

public class LilypondConvert
{
   /**
    * Writes out the sequence to display in the format for lilypond and
    * lilypond-book. Will generate .ly and .html files
    *
    * @param masterS the sequence to write the display for
    * @param header a header for the html file
    * @throws IOException if it cannot write to a .html or .ly file
    */
   public static void writeFiles(Sequence masterS, String header)
      throws IOException
   {
      FileWriter fstreamHTML = new FileWriter("test.html");
      FileWriter fstream = new FileWriter("test.ly");
      BufferedWriter outHTML = new BufferedWriter(fstreamHTML);
      BufferedWriter out = new BufferedWriter(fstream);
      outHTML
         .write("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\">\n"
            + "<!-- header_tag -->\n <html>\n <body> \n ");
      outHTML.write(header);
      outHTML.write("\n<lilypond>\n");
      out.write("\\score \n { \n\\new Staff \n<< \n\\new Voice\n {\n");
      outHTML.write("\\score \n { \n\\new Staff \n<< \n\\new Voice\n {\n");
      for (int i = 0; i < masterS.returnSize(); i++)
      {
         out.write(masterS.chordAt(i).write());
         outHTML.write(masterS.chordAt(i).write());
      }
      outHTML
         .write("\n}\n>>\n\\layout {}\n\\midi {\n\t\\context {\n\t\\Score\n\ttempoWholesPerMinute = #(ly:make-moment 100 4) \n}\n}\n}"
            + "   \n\\version \"2.12.3\"");
      out.write("\n}\n>>\n\\layout {}\n\\midi {\n\t\\context {\n\t\
\Score\n\ttempoWholesPerMinute = #(ly:make-moment 100 4) \n}\n}\n}"
         + "   \n\\version \"2.12.3\"");
      outHTML.write("\n</lilypond>\n");
      outHTML
         .write("<embed src=\"../test.midi\" width=\"140\" height=
\"40\" autostart=\"false\" loop=\"FALSE\"> </embed>");
      outHTML.write("\n\n</body>\n</html>");
      out.close();
      outHTML.close();
   }

   //*Irene*//

   /**
    * Executes a script to call on the 'lilypond' and 'lilypond-book'
program
    * lilypond: converts a .ly file into a .midi .pdf and .ps file
    * lilypond-book: converts a .html file without the graphics to
a .html with
    * the graphic sheet music
    *
    * @throws IOException
    */
   public static void executeLilypond() throws IOException
   {
      Process p = Runtime.getRuntime().exec("./lilypond.sh");
      BufferedReader stdInput = new BufferedReader(new
InputStreamReader(
         p.getInputStream()));
      BufferedReader stdError = new BufferedReader(new
InputStreamReader(
         p.getErrorStream()));
      String s = null;
      while ((s = stdInput.readLine()) != null)
      {
         System.out.println(s);
      }
      while ((s = stdError.readLine()) != null)
      {
            System.out.println(s);
        }
        System.out.println("Done!");
    }
}

```

Other files

```bash
#!/bin/bash
#//*Irene*//

rm test.midi
rm test.pdf
rm test.ps

/Applications/LilyPond.app/Contents/Resources/bin/lilypond test.ly

rm -fr out

/Applications/LilyPond.app/Contents/Resources/bin/lilypond-book --output=out test.html

#open test.pdf
open out/test.html
```

Extra code not used

```
package lexer;

import java_cup.runtime.*;
import lilypond.*;
import java.util.ArrayList;


action code {:

    ParseTree tree = new ParseTree() ;

 :};

parser code {:

:};
    terminal STRING_TEXT, ID, TEST ;
    terminal NEW, ASSIGN;
    terminal COLOR,SHAPE, INSTRUMENT;
    terminal NOTE, CHORD, SEQUENCE;
    terminal COMMA, SEMICOLON, PERIOD, COLON ;

    terminal AND, OR ;
    terminal IF, WHILE ;
    terminal TRUE, FALSE, BOOL ;

    terminal PLUS, MINUS, TIMES, PERCENT, UMINUS ;
    terminal INT, DIGIT;

    terminal LPAREN, RPAREN, LBRACK, RBRACK, LBRACE, RBRACE ;
    terminal LTHAN, GTHAN ;
    terminal LTHANEQ, GTHANEQ ;
    terminal EQUAL, NE, FSLASH ;

    terminal ADD_TO_SEQUENCE, CHANGE_DURATION, FOREACH, DISPLAY ;

non terminal    expr_list, expr_part ;
non terminal    note, chord, sequence ;
non terminal    assign, var ;
non terminal    add_to_sequence, concat_seq, display,
change_duration, chord_in_seq, note_in_seq, sub_seq ;
```

```
non terminal    list_variables, list_notes, list_chords ;

non terminal    rel_expression, eq_expression, boolean_expression, math_expression, int_or_bool_assign;
non terminal    if, while, foreach ;

non terminal    control, object_types, functions ;

precedence left PLUS, MINUS;
precedence left TIMES, FSLASH, PERCENT;
precedence left AND, OR;
precedence left EQUAL, NE;
precedence left UMINUS;


start with expr_list ;

expr_list ::= expr_part expr_list | expr_part ;

expr_part ::=  functions | control | object_types | assign |  var ;

object_types ::= note | chord | sequence ;
control ::= while | if ;
functions ::= add_to_sequence | concat_seq SEMICOLON | display | change_duration | chord_in_seq SEMICOLON | note_in_seq SEMICOLON | sub_seq SEMICOLON ;

note ::= NOTE ID:name ASSIGN NEW NOTE LPAREN ID:pitch COMMA DIGIT:octave COMMA DIGIT:duration RPAREN SEMICOLON
        {:
            tree.NOTE((String) name, (String) pitch, (Integer) octave, (Integer) duration) ;
        :}
        |NOTE ID:name SEMICOLON
        {:
            tree.NOTE((String) name);
        :}
        |ID:name ASSIGN NEW NOTE LPAREN ID:pitch COMMA DIGIT:octave COMMA DIGIT:duration RPAREN SEMICOLON
        {:
            tree.INITIALIZE_NOTE((String) name, (String) pitch,
(Integer) octave, (Integer) duration) ;
        :}
        ;

list_notes ::= ID:name1 COMMA list_notes:name2
        {:
            RESULT = tree.MULT_NOTES((String) name1, (Chord) name2) ;
        :}
        |
        ID:name
        {:
            RESULT = tree.ONE_NOTE((String) name) ;
        :}
        ;

chord ::= CHORD ID:name ASSIGN NEW CHORD LPAREN list_notes:notes RPAREN SEMICOLON
        {:
            tree.CHORD((String) name, (Chord) notes) ;
        :}
        |CHORD ID:name SEMICOLON
        {:
            tree.CHORD((String) name);
        :}
        |ID:name ASSIGN NEW CHORD LPAREN list_notes:notes RPAREN SEMICOLON
        {:
            tree.INITIALIZE_CHORD((String) name, (Chord) notes) ;
        :}
        ;

list_chords ::= DIGIT:name1 COMMA list_chords:name2
    {:
        RESULT = tree.MULT_CHORDS((Integer)name1, (String)name2);
    :}
    |DIGIT:name
    {:
        RESULT = tree.ONE_CHORDS((Integer)name);
    :}
    ;
```

```

sequence ::= SEQUENCE ID:name ASSIGN NEW SEQUENCE LPAREN RPAREN SEMICOLON
    {:
        tree.SEQUENCE((String) name) ;
    :}
    |SEQUENCE ID:name SEMICOLON
    {:
        tree.SEQ((String) name);
    :}
    ;

add_to_sequence ::= ID:sequence ADD_TO_SEQUENCE list_variables:variable_list SEMICOLON
    {:
        tree.ADD_TO_SEQUENCE((String) sequence, (ArrayList<Object>) variable_list) ;
    :}
    ;

chord_in_seq ::= ID:name1 LPAREN DIGIT:num RPAREN
    {:
        RESULT = tree.CHORD_IN_SEQ((String)name1, (Integer) num);
    :}
    ;

note_in_seq ::= ID:name1 LTHAN DIGIT:num_chord COMMA DIGIT:num_note GTHAN
    {:
        RESULT = tree.NOTE_IN_SEQ((String)name1, (Integer) num_chord, (Integer)num_note);
    :}
    ;

sub_seq ::= ID:name LBRACK DIGIT:num1 COLON DIGIT:num2 RBRACK
    {:
        RESULT = tree.SUB_SEQ((String)name,(Integer) num1,(Integer) num2);
    :}
    | ID:name LBRACK list_chords:chordlist RBRACK
    {:
        RESULT = tree.SUB_SET((String)name, (String) chordlist);
    :}
    ;

concat_seq ::= LBRACK ID:s1 COMMA ID:s2 RBRACK
    {:
        RESULT = tree.CONCAT_SEQ((String) s1, (String)s2);
    :}
    ;

list_variables ::= ID:name1 PLUS list_variables:variable_list
        {:
            RESULT = tree.MULT_VARIABLES((String) name1, (ArrayList<Object>) variable_list) ;
        :}
        |
        ID:name
        {:
            RESULT = tree.ONE_VARIABLE((String) name) ;
        :}
        |
        sub_seq:name1 PLUS list_variables:variable_list
        {:
            RESULT = tree.MULT_VARIABLE_N((Object) name1, (ArrayList<Object>) variable_list) ;

        :}
        |
        sub_seq:name
        {:
            RESULT = tree.ONE_VARIABLE_N((Object) name) ;
        :}
        |
        concat_seq:name1 PLUS list_variables:variable_list
        {:
            RESULT = tree.MULT_VARIABLE_N((Object) name1, (ArrayList<Object>) variable_list) ;

        :}
        |
        concat_seq:name
        {:
            RESULT = tree.ONE_VARIABLE_N((Object) name) ;
        :}
        |
        chord_in_seq:name1 PLUS list_variables:variable_list
        {:
            RESULT = tree.MULT_VARIABLE_N((Object) name1, 
(ArrayList<Object>) variable_list) ;

        :}
        |
        chord_in_seq:name
        {:
            RESULT = tree.ONE_VARIABLE_N((Object) name) ;
        :}
        |
        note_in_seq:name1 PLUS list_variables:variable_list
        {:
            RESULT = tree.MULT_VARIABLE_N((Object) name1, 
(ArrayList<Object>) variable_list) ;

        :}
        |
        note_in_seq:name
        {:
            RESULT = tree.ONE_VARIABLE_N((Object) name) ;
        :}
        ;


assign ::= ID:name LTHAN COLOR LPAREN STRING_TEXT:attribute RPAREN 
SEMICOLON
    {:
        tree.ATTRIBUTE_COLOR((String) name, (String) attribute);
    :}
    |ID:name LTHAN INSTRUMENT LPAREN STRING_TEXT:attribute RPAREN 
SEMICOLON
    {:
        tree.ATTRIBUTE_INSTRUMENT((String) name, (String) 
attribute);
    :}
    |ID:name LTHAN SHAPE LPAREN STRING_TEXT:attribute RPAREN SEMICOLON
    {:
        tree.ATTRIBUTE_SHAPE((String) name, (String) attribute);
    :}
    | foreach:name LTHAN COLOR LPAREN STRING_TEXT:attribute RPAREN SEMICOLON
    {:
        tree.FOREACH_COLOR((String) name,(String)attribute);
    :}
    | foreach:name  LTHAN SHAPE LPAREN STRING_TEXT:attribute RPAREN SEMICOLON
    {:
        tree.FOREACH_SHAPE((String) name,(String) attribute);
    :}
    | foreach:name  LTHAN INSTRUMENT LPAREN STRING_TEXT:attribute RPAREN SEMICOLON
    {:
        tree.FOREACH_INSTRUMENT((String) name,(String) attribute);
    :}
    ;



var ::= NOTE ID:name1 ASSIGN ID:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_NOTE((String) name1, (String)name2, true);
    :}
    | CHORD ID:name1 ASSIGN ID:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_CHORD((String) name1, (String)name2, true);
    :}
    | SEQUENCE ID:name1 ASSIGN ID:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_SEQ((String) name1, (String)name2, true);
    :}
    | ID:name1 ASSIGN ID:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR((String) name1, (String)name2);
    :}
    | CHORD ID:name1 ASSIGN chord_in_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_CHORD((String) name1, (Chord) name2, true);
    :}
    | NOTE ID:name1 ASSIGN note_in_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_NOTE((String) name1, (Note)name2, true);
    :}
    | SEQUENCE ID:name1 ASSIGN sub_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_SEQ((String) name1, (Sequence)name2, true);
    :}
    | ID:name1 ASSIGN chord_in_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_CHORD((String) name1, (Chord)name2, false);
    :}
    | ID:name1 ASSIGN note_in_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_NOTE((String) name1, (Note)name2, false);
    :}
    | ID:name1 ASSIGN sub_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_SEQ((String) name1, (Sequence)name2,false);
    :}
    | ID:name1 ASSIGN concat_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_SEQ((String) name1, (Sequence)name2,false);
    :}
    | SEQUENCE ID:name1 ASSIGN concat_seq:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR_SEQ((String) name1, (Sequence)name2, true);
    :}
    | INT ID:name1 ASSIGN math_expression:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR((String) name1, (Integer) name2);
    :}
    | ID:name1 ASSIGN math_expression:name2 SEMICOLON
    {:
        tree.ASSIGN_VAR((String) name1, (Integer) name2);
    :}
    | INT ID:name1 SEMICOLON
    {:
        tree.INT((String) name1);
    :}
    | BOOL ID:name1 ASSIGN boolean_expression:name2 SEMICOLON
    {:
        boolean temp;
        if((Integer) name2==1){
            temp = true;
        }
        else{
            temp = false;
        }

        tree.ASSIGN_VAR((String) name1, (Boolean) temp);
    :}
    | ID:name1 ASSIGN boolean_expression:name2 SEMICOLON
    {:
        boolean temp;
        if((Integer) name2==1){
            temp = true;
        }
        else{
            temp = false;
        }
        tree.ASSIGN_VAR((String) name1, (Boolean) temp);
    :}
    | BOOL ID:name1 SEMICOLON
    {:
        tree.BOOL((String) name1);
    :}
    ;

if ::= IF LPAREN boolean_expression:name1 RPAREN TEST:expressions SEMICOLON
        {:
                if(((Integer) name1) == 1)
                {
                    tree.IF((String) expressions) ;
                }
        :}
        ;

```

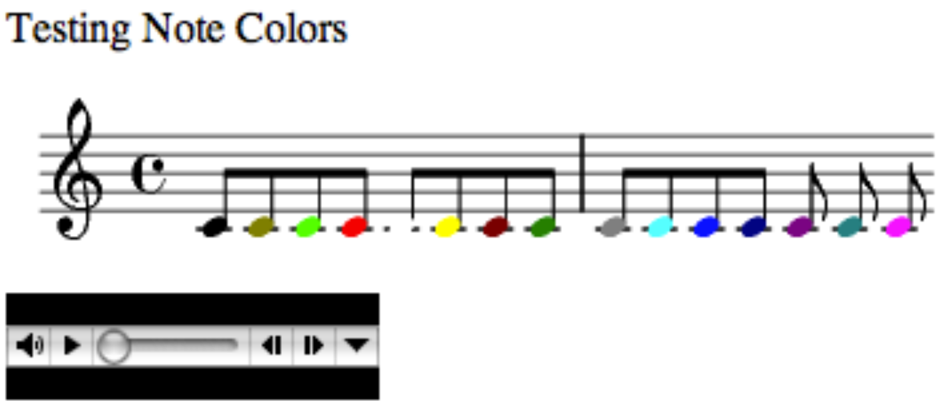
```
while ::= WHILE LPAREN boolean_expression:exp RPAREN
TEST:expressions SEMICOLON
        {:
                tree.WHILE((Integer) exp, (String) expressions) ;
        :}
        ;

foreach ::= FOREACH LPAREN ID:name RPAREN
        {:
                RESULT = name ;
        :}
        ;



change_duration ::= ID:name CHANGE_DURATION DIGIT:num SEMICOLON
    {:
        tree.CHANGE_DURATION((String) name, (Integer) num);
    :}
    ;


display ::= DISPLAY LPAREN ID:name COMMA STRING_TEXT:header RPAREN
SEMICOLON
    {:
        tree.DISPLAY((String) name, (String) header) ;
    :}
    ;


///////////////////////Unused
code /////////////////////////////////////

//*Taylor*//

math_expression ::= math_expression:name1 PLUS math_expression:name2
    {:
        RESULT = (Integer) name1 + (Integer) name2;
        System.out.println("NEW VALUE: "+RESULT);
    :}
    | math_expression:name1 MINUS math_expression:name2
    {:
        RESULT = (Integer) name1 + (Integer) name2;
                System.out.println("NEW VALUE: "+RESULT);
    :}
    | math_expression:name1 TIMES math_expression:name2
    {:
        RESULT = (Integer) name1 * (Integer) name2;
                System.out.println("NEW VALUE: "+RESULT);

    :}
    | math_expression:name1 FSLASH math_expression:name2
    {:
        RESULT = (Integer) name1 / (Integer) name2;
                System.out.println("NEW VALUE: "+RESULT);

    :}
    | math_expression:name1 PERCENT math_expression:name2
    {:
        RESULT = (Integer) name1 % (Integer) name2;
                System.out.println("NEW VALUE: "+RESULT);

    :}
    | MINUS math_expression:name1
    {:
        RESULT = (Integer) (-1* (Integer) name1);
                System.out.println("NEW VALUE: "+RESULT);

    :} %prec UMINUS
    | LPAREN math_expression:name RPAREN
    {:
        RESULT = (Integer) name;
                System.out.println("NEW VALUE: "+RESULT);

    :}
    | DIGIT:name
    {:
        RESULT = (Integer) name;
                System.out.println("NEW VALUE: "+RESULT);

    :};

```

```
rel_expression ::= math_expression:name1 LTHAN math_expression:name2
    {:
        if((Integer)name1 < (Integer)name2)
        {
            RESULT = 1;
        }
        else
        {
            RESULT = 0;
        }
    :}
    | math_expression:name1 GTHAN math_expression:name2
    {:
        if((Integer)name1 > (Integer)name2){
            RESULT = 1;
        }
        else{
            RESULT = 0;
        }
    :}
    | math_expression:name1 LTHANEQ math_expression:name2
    {:
        if((Integer)name1 <= (Integer)name2){
            RESULT = 1;
        }
        else{
            RESULT = 0;
        }
    :}
    | math_expression:name1 GTHANEQ math_expression:name2
    {:
        if((Integer)name1 >= (Integer)name2){
            RESULT = 1;
        }
        else{
            RESULT = 0;
        }
    :};

eq_expression ::= math_expression:name1 EQUAL math_expression:name2
    {:
        if((Integer)name1 == (Integer)name2){
            RESULT = 1;
        }
        else{
            RESULT = 0;
        }
    :}
    | math_expression:name1 NE math_expression:name2
    {:
        if((Integer)name1 != (Integer)name2){
            RESULT = 1;
        }
        else{
            RESULT = 0;
        }
    :}
    ;

boolean_expression::= eq_expression:name
    {:
        RESULT = (Integer) name;
        System.out.println("NEW BOOLEAN: " + RESULT);
    :}
    | boolean_expression:name1 EQUAL boolean_expression:name2
    {:
        if((Integer)name1 == (Integer) name2){
            RESULT = 1;
        }
        else{
            RESULT = 0;
        }
    :}
    | boolean_expression:name1 NE boolean_expression:name2
    {:
        if((Integer) name1 != (Integer) name2){
            RESULT = 1;
        }
        else{
            RESULT = 0;
        }
    :}
    | rel_expression:name
    {:
        RESULT = (Integer) name;
                System.out.println("NEW BOOLEAN: " + RESULT);

    :}
    | boolean_expression:name1 AND boolean_expression:name2
    {:
        if((Integer)name1==1 && (Integer)name2==1){
            RESULT = 1;
        }
        else{
            RESULT = 0;
        }
                System.out.println("NEW BOOLEAN: " + RESULT);

    :}
    | boolean_expression:name1 OR boolean_expression:name2
    {:
        if((Integer)name1==1 || (Integer)name2==1){
            RESULT = 1;
        }
        else{
            RESULT = 0;
        }
                System.out.println("NEW BOOLEAN: " + RESULT);

    :}
    | LPAREN boolean_expression:name1 RPAREN
    {:
        RESULT = name1;
    :}
    | TRUE
    {:
        RESULT = 1;
                System.out.println("NEW BOOLEAN: " + RESULT);

    :}
    | FALSE{:
        RESULT = 0;
                System.out.println("NEW BOOLEAN: " + RESULT);

    :};



```

Test files

```
//*Farbound*//
//a note can have a color attribute
Note note1 = new Note(C, 4, 8);
Sequence sequence = new Sequence() ;

//is supported by the following colors
note1 < Color('black');
sequence << note1 ;
note1 < Color('darkyellow');
sequence << note1 ;
note1 < Color('green');
sequence << note1 ;
note1 < Color('red');
sequence << note1 ;
note1 < Color('white');
sequence << note1 ;
note1 < Color('yellow');
sequence << note1 ;
note1 < Color('darkred');
sequence << note1 ;
note1 < Color('darkgreen');
sequence << note1 ;
note1 < Color('grey');
sequence << note1 ;
note1 < Color('cyan');
sequence << note1 ;
note1 < Color('blue');
sequence << note1 ;
note1 < Color('darkblue');
sequence << note1 ;
note1 < Color('darkmagenta');
sequence << note1 ;
note1 < Color('darkcyan');
sequence << note1 ;
note1 < Color('magenta');
sequence << note1 ;


display(sequence, "Testing Note Colors") ;

```

Output


Testing Note Colors

```
//*Farbound*//
//a note can be played by different instruments
Note note1 = new Note(C, 4, 4);
Sequence sequence = new Sequence() ;

note1 < Instrument('fx 8 (sci-fi)');
sequence << note1 ;
note1 < Instrument('rock organ');
sequence << note1 ;
note1 < Instrument('brass section');
sequence << note1 ;
note1 < Instrument('sitar');
sequence << note1 ;
note1 < Instrument('church organ');
sequence << note1 ;
note1 < Instrument('synthbrass 1');
sequence << note1 ;
note1 < Instrument('banjo');
sequence << note1 ;
note1 < Instrument('reed organ');
sequence << note1 ;
note1 < Instrument('synthbrass 2');
sequence << note1 ;
note1 < Instrument('shamisen');
sequence << note1 ;
note1 < Instrument('accordion');
sequence << note1 ;
note1 < Instrument('soprano sax');
sequence << note1 ;
note1 < Instrument('koto');
sequence << note1 ;
note1 < Instrument('harmonica');
sequence << note1 ;
note1 < Instrument('alto sax');
sequence << note1 ;
note1 < Instrument('kalimba');
sequence << note1 ;
note1 < Instrument('concertina');
sequence << note1 ;
note1 < Instrument('tenor sax');
sequence << note1 ;
```

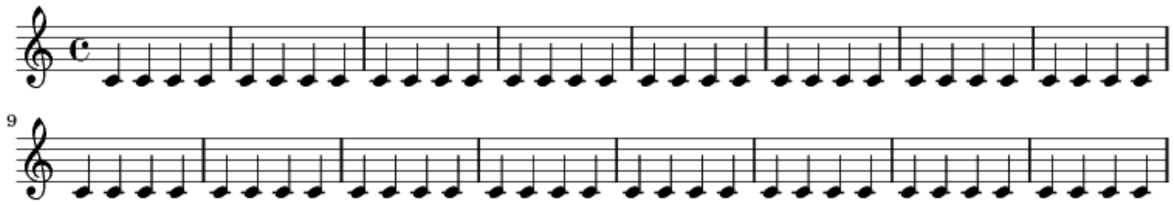
```
note1 < Instrument('bagpipe');
sequence << note1 ;
note1 < Instrument('acoustic guitar (nylon)');
sequence << note1 ;
note1 < Instrument('baritone sax');
sequence << note1 ;
note1 < Instrument('fiddle');
sequence << note1 ;
note1 < Instrument('acoustic guitar (steel)');
sequence << note1 ;
note1 < Instrument('oboe');
sequence << note1 ;
note1 < Instrument('shanai');
sequence << note1 ;
note1 < Instrument('electric guitar (jazz)');
sequence << note1 ;
note1 < Instrument('english horn');
sequence << note1 ;
note1 < Instrument('tinkle bell');
sequence << note1 ;
note1 < Instrument('electric guitar (clean)');
sequence << note1 ;
note1 < Instrument('bassoon');
sequence << note1 ;
note1 < Instrument('agogo');
sequence << note1 ;
note1 < Instrument('electric guitar (muted)');
sequence << note1 ;
note1 < Instrument('acoustic bass');
sequence << note1 ;
note1 < Instrument('pan flute');
sequence << note1 ;
note1 < Instrument('synth drum');
sequence << note1 ;
note1 < Instrument('electric bass (finger)');
sequence << note1 ;
note1 < Instrument('blown bottle');
sequence << note1 ;
note1 < Instrument('reverse cymbal');
sequence << note1 ;
note1 < Instrument('electric bass (pick)');
```

```
sequence << note1 ;
note1 < Instrument('shakuhachi');
sequence << note1 ;
note1 < Instrument('guitar fret noise');
sequence << note1 ;
note1 < Instrument('fretless bass');
sequence << note1 ;
note1 < Instrument('whistle');
sequence << note1 ;
note1 < Instrument('breath noise');
sequence << note1 ;
note1 < Instrument('slap bass 1');
sequence << note1 ;
note1 < Instrument('ocarina');
sequence << note1 ;
note1 < Instrument('seashore');
sequence << note1 ;
note1 < Instrument('slap bass 2');
sequence << note1 ;
note1 < Instrument('lead 1 (square)');
sequence << note1 ;
note1 < Instrument('bird tweet');
sequence << note1 ;
note1 < Instrument('synth bass 1');
sequence << note1 ;
note1 < Instrument('lead 2 (sawtooth)');
sequence << note1 ;
note1 < Instrument('telephone ring');
sequence << note1 ;
note1 < Instrument('synth bass 2');
sequence << note1 ;
note1 < Instrument('lead 3 (calliope)');
sequence << note1 ;
note1 < Instrument('helicopter');
sequence << note1 ;
note1 < Instrument('violin');
sequence << note1 ;
note1 < Instrument('lead 4 (chiff)');
sequence << note1 ;
note1 < Instrument('applause');
sequence << note1 ;
```

```
note1 < Instrument('viola');
sequence << note1 ;
note1 < Instrument('lead 5 (charang)');
sequence << note1 ;
note1 < Instrument('gunshot');
sequence << note1 ;
note1 < Instrument('cello');
sequence << note1 ;
note1 < Instrument('lead 6 (voice)');
sequence << note1 ;

display(sequence, "Testing Note Instruments") ;

```

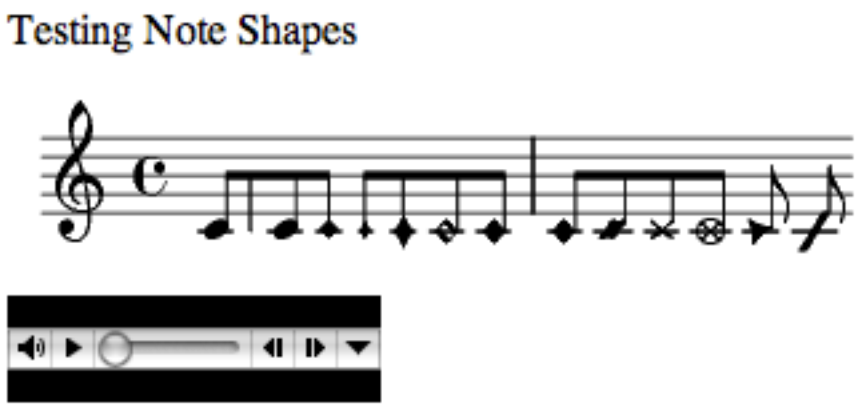
Output

Testing Note Instruments



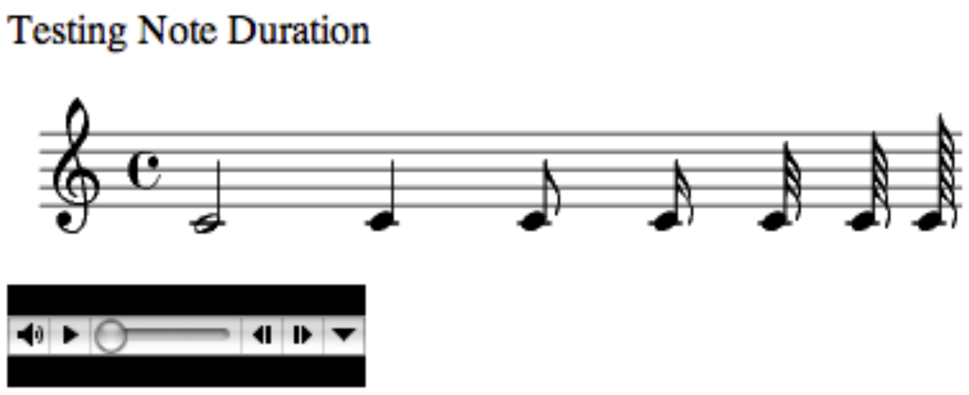
```
//*Farbound*//
//a note can have different shapes
Note note1 = new Note(C, 4, 8);
Sequence sequence = new Sequence() ;

//with the following shapes
note1 < Shape('default');
sequence << note1 ;
note1 < Shape('altdefault');
sequence << note1 ;
note1 < Shape('baroque');
sequence << note1 ;
note1 < Shape('neomensural');
sequence << note1 ;
note1 < Shape('mensural');
sequence << note1 ;
note1 < Shape('petrucci');
sequence << note1 ;
note1 < Shape('harmonic');
sequence << note1 ;
note1 < Shape('harmonic-black');
sequence << note1 ;
note1 < Shape('harmonic-mixed');
sequence << note1 ;
note1 < Shape('diamond');
sequence << note1 ;
note1 < Shape('cross');
sequence << note1 ;
note1 < Shape('xcircle');
sequence << note1 ;
note1 < Shape('triangle');
sequence << note1 ;
note1 < Shape('slash');
sequence << note1 ;

display(sequence, "Testing Note Shapes") ;

```

Output


Testing Note Shapes

```
//*Farbound*//
//a note can have different durations
Note note1 = new Note(C, 4, 2);
Sequence sequence = new Sequence() ;

//and their durations can be modified to the following duration
note1^2;
sequence << note1 ;
note1^4;
sequence << note1 ;
note1^8;
sequence << note1 ;
note1^16;
sequence << note1 ;
note1^32;
sequence << note1 ;
note1^64;
sequence << note1 ;
note1^128;
sequence << note1 ;


display(sequence, "Testing Note Duration") ;


```

Output


Testing Note Duration

```
//*Jonathan*//

Note n1;
Chord c1; // Chord c1 declared but not initialized.
Note n2 = n1; //Error: Variable n1 has not been initialized
n3 = new Note(D, 4, 2);
Note n4 = new Note(D, 4, 2);
Chord c2 = new Chord(n1); //Error: Variable n1 has not been
initialized.
Chord c3 = new Chord(n4);
c3^4; //Error: The ^ operator isn't defined for type lilypond.Chord
c3<Color('darkcyan'); //Error: The < operator isn't defined for
type lilypond.Chord

```

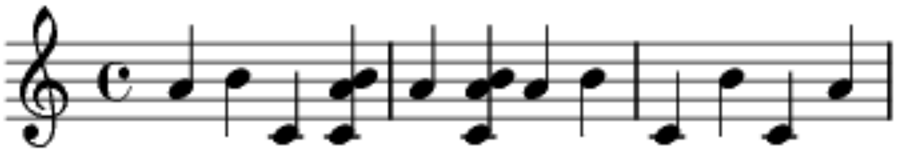
Output

```
Error: Variable n1 has not been initialized.
Error: Variable n3 has not been declared.
Error: Variable n1 has not been initialized.
Error: The ^ operator isn't defined for type lilypond.Chord.
Error: The < operator isn't defined for type lilypond.Chord.
```

```
//*Richard*//

//build up of initial variable
Sequence seq = new Sequence();
Note n1 = new Note(A,4,4);
Note n2 = new Note(B,4,4);
Note n3 = new Note(C,4,4);
Chord c1 = new Chord(n1,n2,n3);

//building an initial sequence
seq<<n1+n2+n3+c1;

//Adds n1 and n3 to the sequence again;
seq<< seq(0) + seq(3);


//Adds the first through 3 chords to sequence again
seq<<seq[0:2];



//Creates a sequence chords at location 1, 2, 4
Sequence newSeq = new Sequence();
newSeq<<seq[1,2,4];


//Concatenates newSeq and seq into finalSeq
Sequence finalSeq = new Sequence();
finalSeq << [seq,newSeq];

display(finalSeq, "Testing built-in functions");
```

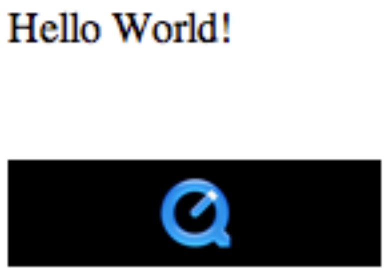
Output

Testing built-in functions



```
//*Irene*//

Sequence sequence1 = new Sequence() ;
display(sequence1, "Hello World!") ;
```

Output



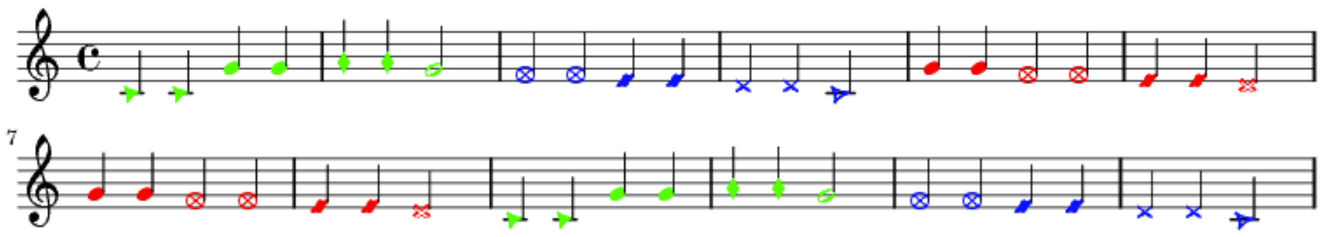
Hello World!

```
//*Richard*//

//Twinkle Twinkle Little Star

Note C = new Note(C, 4, 4) ;
C<Shape('triangle');

Note D = new Note(D, 4, 4) ;
D<Shape('cross');

Note E = new Note(E, 4, 4) ;
E<Shape('diamond');

Note F = new Note(F, 4, 4) ;
F<Shape('xcircle');

Note G = new Note(G, 4, 4) ;
G<Shape('baroque');

Note A = new Note(A, 4, 4) ;
A<Shape('petrucci');

Note Ghalf = new Note(G, 4, 2);
Ghalf<Shape('baroque');

Note Chalf = new Note(C, 4, 2);
Chalf<Shape('triangle');

Note Dhalf = new Note(D, 4, 2);
Dhalf<Shape('cross');

Sequence sequence1 = new Sequence();
sequence1 << C + C + G + G + A + A + Ghalf;
foreach(sequence1)<Color('green');

Sequence sequence2 = new Sequence();
sequence2 << F + F + E + E + D + D + Chalf;
foreach(sequence2)<Color('blue');

Sequence sequence3 = new Sequence();
sequence3 << G + G + F + F + E + E + Dhalf;
```

```
foreach(sequence3)<Color('red');

Sequence everything = new Sequence();
everything<<sequence1 + sequence2 + sequence3 + sequence3 +
sequence1+ sequence2;


display(everything, "Twinkle Twinkle Little *") ;
```

Output

Twinkle Twinkle Little *



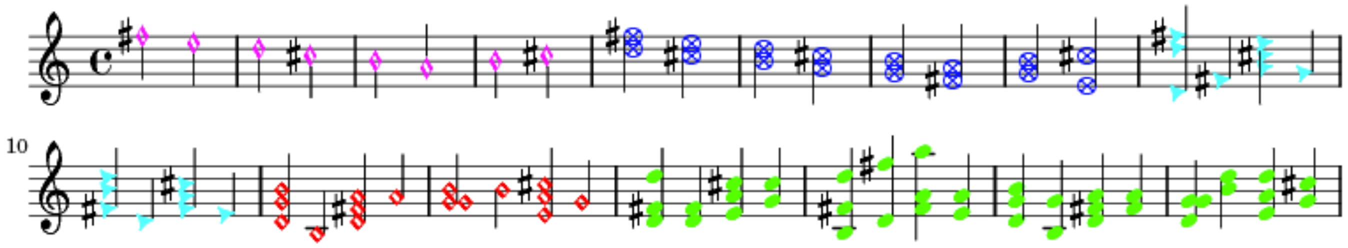
```
//*Irene*//

Note A2 = new Note(A, 5, 4) ;
Note F2 = new Note(F#, 5, 2) ;
Note E2 = new Note(E, 5, 2) ;
Note D2 = new Note(D, 5, 2) ;
Note C2 = new Note(C#, 5, 2) ;
Note B2 = new Note(B, 4, 2) ;
Note A = new Note(A, 4, 2) ;
Note G = new Note(G, 4, 2) ;
Note F = new Note(F#, 4, 2) ;
Note E = new Note(E, 4, 2) ;
Note D = new Note(D, 4, 4);
Note B = new Note(B, 3, 4);

Sequence s1 = new Sequence() ;
s1 << F2 + E2 + D2 + C2 + B2 + A + B2 + C2 ;

foreach(s1) < Color('magenta') ;
foreach(s1) < Instrument('violin') ;
foreach(s1) < Shape('petrucci');

Sequence s2 = new Sequence() ;


Chord c1 = new Chord(D2, F2) ;
Chord c2 = new Chord(C2, E2) ;
Chord c3 = new Chord(B2, D2) ;
Chord c4 = new Chord(A, C2) ;
Chord c5 = new Chord(G, B2) ;
Chord c6 = new Chord(F, A) ;
Chord c7 = new Chord(G, B2) ;
Chord c8 = new Chord(E, C2) ;

s2 << c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8 ;

F2^4 ;
E2^4 ;
D2^4 ;
C2^4 ;
B2^4 ;
```

```
A^4 ;
G^4 ;
F^4 ;
E^4 ;

foreach(s2) < Color('blue') ;
foreach(s2) < Instrument('rock organ') ;
foreach(s2) < Shape('xcircle') ;

Sequence s3 = new Sequence() ;


c1 = new Chord(D, D2, F2) ;
c2 = new Chord(F) ;
c3 = new Chord(A, C2, E2) ;
c4 = new Chord(G) ;
c5 = new Chord(F, B2, D2) ;
c6 = new Chord(D) ;
c7 = new Chord(F, A, C2) ;
c8 = new Chord(E) ;

s3 << c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8 ;

c1 = new Chord(D, G, B2) ;
c2 = new Chord(B) ;
c3 = new Chord(D, F, A) ;
c4 = new Chord(A) ;
c5 = new Chord(G, G, B2) ;
c6 = new Chord(B2) ;
c7 = new Chord(A, E, C2) ;
c8 = new Chord(G) ;

foreach(s3) < Color('cyan') ;
foreach(s3) < Instrument('bagpipe') ;
foreach(s3) < Shape('triangle') ;


Sequence s4 = new Sequence() ;

s4 << c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8 ;

```

```
c1 = new Chord(F, D, D2) ;
c2 = new Chord(D, F) ;
c3 = new Chord(E, A, C2) ;
c4 = new Chord(C2, G) ;
c5 = new Chord(D2, F, B) ;
c6 = new Chord(F2, D) ;
c7 = new Chord(A2, F, A) ;
c8 = new Chord(A, E) ;


foreach(s4) < Color('red') ;
foreach(s4) < Instrument('soprano sax') ;
foreach(s4) < Shape('harmonic') ;


Sequence s5 = new Sequence() ;

s5 << c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8 ;


c1 = new Chord(B2, D, G) ;
c2 = new Chord(G, B) ;
c3 = new Chord(A, D, F) ;
c4 = new Chord(F, A) ;
c5 = new Chord(D, G, G) ;
c6 = new Chord(D2, B2) ;
c7 = new Chord(D2, A, E) ;
c8 = new Chord(C2, G) ;

s5 << c1 + c2 + c3 + c4 + c5 + c6 + c7 + c8 ;
foreach(s5) < Color('green') ;
foreach(s5) < Instrument('acoustic grand');

Sequence s6 = new Sequence() ;

s6 << s1 + s2 + s3 + s4 + s5;

display(s6, "Pachelbel Canon") ;
```

Output

Pachelbel Canon



```
//*Irene*//

Note n1 = new Note(E, 4, 4) ;
Note n2 = new Note(G, 4, 4) ;
Note n3 = new Note(B, 4, 4) ;
Note n4 = new Note(D, 5, 4) ;
Note n5 = new Note(F, 5, 4) ;


//A
Sequence s1 = new Sequence() ;

Chord a1 = new Chord(n1,n2,n3,n4);
Chord a2 = new Chord(n3,n5);

s1 << a1 + a2 + a2 + a1 ;

foreach(s1) < Color('red');
foreach(s1) < Shape('xcircle');

//H
Sequence s2 = new Sequence() ;

Chord h1 = new Chord(n1,n2,n3,n4,n5) ;
Chord h2 = new Chord(n3) ;

s2 << h1 + h2 + h2+ h1 ;

foreach(s2) < Color('black');
foreach(s2) < Shape('harmonic-black');


//O
Sequence s3 = new Sequence() ;

Chord o1 = new Chord(n2,n3,n4) ;
Chord o2 = new Chord(n1,n5);

s3 << o1 + o2 + o2 + o1 ;

foreach(s3) < Color('grey');
```

```

//

Sequence s4 = new Sequence() ;
s4 << s1 + s2 + s3 ;


display(s4, "AHO") ;
```

Output

AHO