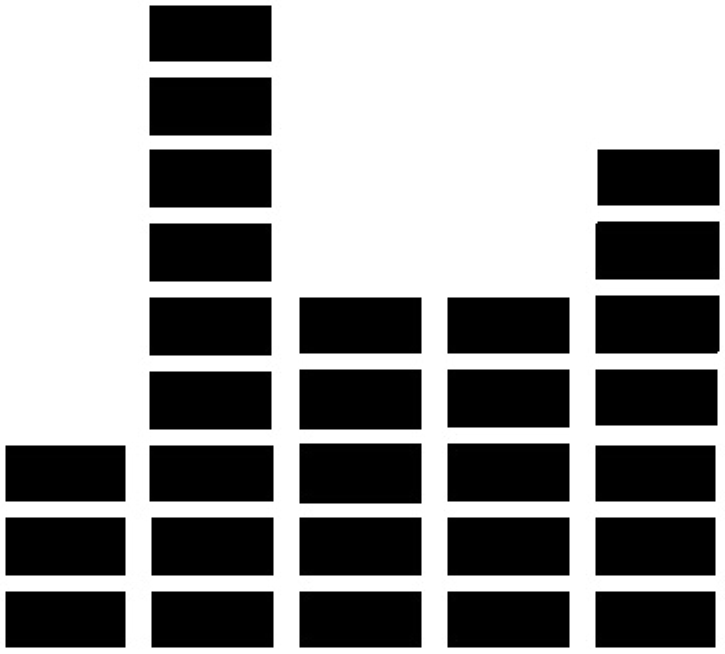
COMS 4115 Programming Languages and Translators
Fall 2013
Professor Edwards

Lullabyte Final Report



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

Table of Contents

1. Introduction

1.1. Overview
Lullabyte is a programmatic abstraction of music composition, utilizing syntax similar to C and C-derived languages. With Lullabyte, developers can construct basic sounds, string together series of sounds to form musical tracks, and layer multiple tracks to produce complex compositions. Lullabyte programs produce bytecode that is converted to MIDI files utilizing the open-source Java API, JFugue.

1.2 Motivation

-Composition-
Lullabyte allows developers to create musical compositions, either directly or algorithmically. Developers can create music through programs that directly manipulate a song's common elements (pitch, melody, rhythm, etc.). In this manner, composition is essentially the same as traditional methods. Alternatively, users can develop more complicated compositions by writing programs that generate melodies automatically. In this case, repetitive elements can be generated or iterated on much more easily than traditional methods.

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

1.3. Related Work

ChucK	Concurrent, strongly timed audio programming language for real-time synthesis, composition, and performance.
jMusic	Has a data structure that is based on a musical score metaphor, and consists of a hierarchy of notes, phrases, parts and score.
Csound	Takes two specially formatted text files as input. The *orchestra* describes the nature of the instruments and the *score* describes notes and other parameters along a timeline.
Pure Data	A visual programming language for creating interactive computer music.

2. Language Tutorial

2.1. Variables

Lullabyte contains five basic variable types: int, double, boolean, pitch, and sound. A sixth type, void, can be specified as a return type for a function. Within a given scope, all variables must be declared before any variable is assigned a value.

```
int i_loc;                 // i_loc declared with default value 0;
double d_loc;              // d_loc declared with default value 0.0;
boolean b_loc;             // b_loc declared with default value false;
pitch p_loc;               // p_loc declared with default value C0;
sound s_loc;               // s_loc declared with default value |C0|:0.:0

i_loc = 2;                 // i_loc assigned value of 2;
d_loc = 2.0;               // d_loc assigned value of 2.0;
b_loc = true;              // b_loc assigned value of true;
p_loc = A2;                // p_loc assigned value of A2;
s_loc = |A2|:0.125:50;     // s_loc assigned value of |A2|:0.125:50;
```

2.2. Arrays

An array is a data structure that stores one or more elements of the same type. An array's type determines the category of object held in each of its cells, and is determined at declaration. Setting an array element's value at an index beyond the current capacity grows the array and fills intermediary values with appropriate placeholders, based on its type.

```
boolean[] b;                      // b = [false];
b[4] = true;                      // b = [false, false, false, true];
b = [true, false, true];          // b = [true, false, true];
```

2.3. Control Flow

Statements in source files generally execute from top to bottom, starting in main. Control flow statements, however, break up the flow of execution by employing decision making and looping. Lullabyte employs five types of control flow statements: if-then, if-then-esle, for, while, and loop.

```
int i;
int[] int_array;

/* if-then statement */
if (true) {
      func1();                            // func1() will execute
}


/* if-then-else statement */
if (false) {
      func1();                            // func1() will not execute
} else {
      func2();                            // func2() will execute
}


/* for statement */
for (i = 0 ; i < 5 ; i = i + 1) {
      func();                             // func() will execute 5 times
}


/* while statement */
while(i < 20) {
      func();                             // func() will execute 15 more times
      i = i + 1;
}

/* loop statement */
int_array = [1, 2, 3];
loop(number : int_array) {
      number = number * 3;                // int_array = [3, 6, 9]
}
```

2.4. Functions

Functions are a collection of statements, defined outside the main function and given a name and optional return type. Functions can accept zero or more argument variables of any type and operate on argument values and global variables. Functions that do not return a value should specify a return type of void.

```
/* function with no return type or arguments*/
void incrementGlobal(){
      global_int = global_int + 1;
}



/* recursive function, taking an argument and returning a value */
int factorial(int n){
      if (n<=2){ return n; }
      else { return n * factorial(n-1); }
}
```

2.5. Program Structure

As in C, a program in Lullabyte only requires a main function. However, typical Lullabyte programs consist of some global variable declarations, function definitions, and a main function in which mixDown is called one or more times.

```
/* global variable declarations */
sound[] global_sounds;



/* function definitions*/
void func1(){...}
int func2(double d, boolean b){...}
sound[] func3(sound[] arg){...}



/* required main function */
void main(){
      // local variable declaration
      sound[] local_sounds; int i;



      // function calls
      func1();
      i = func2(5.5, false);
      local_sounds = func3([|G5|:0.25:100, |A5|:0.25:100, |B5|:0.25:100]);

      // midi file construction
      bpm(80);
      mixDown(global_sounds, 1);
      mixDown(local_sounds, 2);
}
```

2.6. Using the compiler

The first time using Lullabyte you must compile the project. To do this run, make:

```
$ make
```

To compile Lullabyte files, run lullabyte, which takes in accepts an .llb file and an optional file name for the midi file:

```
   lullabtye [llbFile] [MidiFileName-optional]
$ ./lullabtye MyMusic.llb MyMusicMidi.mid
```

Lullabyte will run the .llb file (MyMusic.llb above) and generate bytecode, which will be used to generate a MIDI file. If the .llb file fails for some reason, the bytecode will contain only "x n\", signaling the compiler not to generate a MIDI file. Here is an example of a simple two track bytecode, generated by calling mixDown three times:

```
120 b
0[[D5, B5, G5]:0.25:100,[C5, A5, F5]:0.25:100,[E5, C5, G5]:0.25:100]
2[[D6]:0.083:100,[D6]:0.083:100,[D6]:0.083:100,[D6]:0.25:100,[A#6, F6]:0.25:100,[E6,
C6]:0.25:100,[F6, D6]:0.083:100,[D6]:0.083:0,[E6, C6]:0.083:100,[E6, D6]:0.75:100]
0[[D5, B5, G5]:0.25:100,[C5, A5, F5]:0.25:100,[E5, C5, G5]:0.25:100]
```

The number 120 indicates the beats per minute. The character "b" indicates that the process should both generate a MIDI file and play it. The numbers before the arrays - 0 and 2 - represent the two track numbers. Tracks are played at the same time and the same track numbers will be appended together. The arrays are the sequences of chords for their track specified.

If the bytecode isn't empty, lullabyte will then call BytecodeTranslator.java and generate a MIDI file. By default, lullabyte will save the resulting MIDI file and play it. Happy listening!

2.7. Examples

2.7.1. Fibonacci Sequence

Below is an example Lullabyte program, which computes the Fibonacci sequence to a given integer, and uses the resulting list of numbers to create a sequence of notes whose distance from the starting pitch is their index in the Fibonacci sequence.

```
int fib (int n){
     if (n <= 2){
            return 1;
     } else {
            return fib(n - 1) + fib(n - 2);
     }
}

sound[] offsetFib(pitch p){
     int offset;
     sound[] accumulator;
     sound s;
     int i;
     accumulator = [];



     s = setAmplitude(s, 75);
     s = setDuration(s, 0.25);



     for (i = 1; i < 11; i = i + 1){
            offset = fib(i);
            s = setPitches(s, [p + offset]);
            accumulator[i-1] = s;
     }
     return accumulator;
}

void main(){
     pitch pitchC2;
     sound[] fibScale;
     pitchC2 = C2;
     fibScale = offsetFib(pitchC2);
     print(fibScale);
     mixDown(fibScale);
}
```

The first function definition in the program computes the Fibonacci sequence to a given number of integers. The second function definition does something more unique to our language: given some pitch p, it calculates 11 numbers in the Fibonacci sequence and uses these numbers to build up a sound array. There are a few things to take away from this function declaration. Every variable being used in the function must be declared at the top of the function. The next lines demonstrate the setAmplitude and setDuration built-in functions. Each function takes two arguments and returns a sound object. setAmplitude requires a sound to be modified and an expression that evaluates to an integer. setDuration requires a sound to be modified and an expression that evaluates to a double. Since sound objects cannot be manipulated directly after creation, these functions are needed to create a sound with the attributes you want.

The next statement is a for-loop, which will look familiar if you've ever used a language with C-like syntax. Because variables must be declared at the beginning of the function, the scope of the index persists after the end of the loop.

Inside of the loop is where most of the work happens. We get the offset for a given index using the fib function. Then, we modify our sound 's' that was defined above. Another built-in function is used, setPitches. This function takes a sound and an array of pitches as an argument. The sound 's' is used as input because we want out new sound to have the same duration and amplitude as s. The second argument is our input argument 'pitch p', added to an integer. This returns a pitch which is offset semitones higher than the pitch p. We then create an array from this value, because setPitches takes an array of pitches as its second argument. Overall, this line has the effect of incrementing our base pitch p by a number in the Fibonacci sequence.

The next line adds our new sound to the accumulator sound array. Notice that the array bounds have not been set; the array grows to the length of the greatest indexed value assignment. This can be a potentially slow operation, so if your focus is on the performance of your program (which it probably isn't if you're writing Lullabyte code), consider pre-allocating the array by setting a temporary value to the last index you will use.

The main function is simple. It initializes values that will be used later in the function. Here is where we set our first pitch, C2. This value corresponds to a low frequency note at 65.406Hz, according to the MIDI specifications. We then pass C2 to our function, which calculates the fibScale sound array. We then print the array using built-in function print. Lastly, we use the built-in mixDown function to create an output MIDI file, which is the last event in our program.

2.7.2. Blues Scale

```
sound[] randomBluesArray(pitch p, int n, double d){
      sound[] acc; int[] blues; sound s; int i; int r; pitch newPitch;
      s = setPitches(s, [p]);
      s = setDuration(s, d);
      s = setAmplitude(s, 75);
      blues = [0, 3, 5, 6, 7, 10];
      for (i = 0; i < n; i = i + 1){
              r = randomInt(4);
              newPitch = p + blues[r];
              s = setPitches(s, [newPitch]);
              acc[i] = s;
      }
      return acc;
}
void main() {
      sound[] blues; sound b;
      blues = randomBluesArray(C4, 20, 0.125);
      mixDown(blues, 0);

      loop(b : blues) {
              b = setDuration(b, 0.125);
      }
      mixDown(blues, 0);
}
```

This program demonstrates the versatility and algorithmic capabilities of our language. Here, we are generating a series of sounds based on a base pitch, p, and an array of offset values that would produce a blues sequence. This is done in the randomBluesArray method which uses a for loop to build each element in the sound array acc. The new pitch is generated by selecting a random offset value in blues[] and offsetting the base pitch p. The new pitch is then set to the sound variable s which is then assigned to the current element in acc. Once the array is built, the for loop terminates and the newly generated sound array acc is returned.

In main, the random blues array that was generated is stored in a sound array, blues, which is then written to the byte code file with the mixDown function. After, we demonstrate Lullabyte's loop statement. Here, it iterates through each element in the array, blues, and makes b the reference to the current element. In this case, b is being passed as an argument into the setDuration standard library function and being assigned to, which means each array element's value will be accessed and overwritten. In this case, the duration of each sound is simply being changed. The loop syntax makes it easy for the programmer to iterate through arrays and make modifications to them. Lastly, the new blues array is written to the mixDown function for the second time which means the new set of sounds will be appended to the existing bytecode file.

The randomBluesArray method could be expanded in many ways using Lullabyte. The arguments could be modified to allow for arbitrary scales, or scales with probability values associated with each note. The method could also take in an array of pitches and grab only notes in the pitch array.

This example shows how sounds and note sequences of significant variation and complexity as a blues sequence can be generated with relatively simple and minimal code in Lullabyte.

3. Language Reference Manual

3.1. Introduction

Lullabyte is an intuitive and robust programmatic abstraction of music composition, utilizing syntax similar to C and C-derived languages. This manual describes, in detail, the lexical conventions, data types, expressions, statements, functions, rules, file format, scope, and output of the Lullabyte programming language.

3.2. Lexical Conventions

As in C, a program in Lullabyte only requires a main function. However, typical Lullabyte programs are more complex, including any number of global and local variable declarations, variable assignments, function definitions, function calls, and operations.

3.2.1. Comments

Single-line comments are initiated with two slash characters (//) and tell the compiler to ignore all content until the end of the current line.

```
// this is a comment
```

Multi-line comments are initiated with a slash and star character (/*) and terminated with a star and slash character (*/); the compiler ignores all content between the indicators. This type of comment does not nest.

```
/* this is a comment */
```

3.2.2. White Space

Spaces, tabs, and newline characters are all white space characters. White space refers to any group of one or more white space characters in series. White space is ignored in all cases except when used to separate adjacent identifiers, keywords, literals, and constants.

3.2.3. Identifiers

An identifier consists of a sequence of letters and digits. An identifier must start with a letter. A new valid identifier cannot be the same as reserved keywords or pitch literals (see Keywords and Literals). An identifier has no strict limit on length and can be composed of the following characters:

```
a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z
A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z
0  1  2  3  4  5  6  7  8  9  _
```

3.2.4. Keywords

Keywords are reserved identifiers with special meanings. They are used for interaction with external packages or act as type Identifiers. The following keywords are reserved:

```
main                else                void                pitch
return              while               boolean             sound
function            true                int
if                  false               double
```

3.2.5. Constants

Integer Constants
An integer constant is a sequence of digits. 123 and 0 are examples of integer. On declaration, integers initialize to a default to a value of 0.

Double Constants
A double constant consists of an integer part, followed by a decimal and a fraction part, written in decimal form. When assigning a value to a double, one or both of the integer part and fraction part must be present. On declaration, doubles initialize to a default to a value of 0.0.

Boolean Constants
A boolean constant is a binary variable with a value of either true or false. On declaration, booleans initialize to a default to a value of false.

Pitch Constants
Pitch constants epitomize conventional representation of pitches: a capitalized character from ‘A’ to ‘G’ followed by an optional ‘#’ or ‘b’ and a single digit integer from 0 to 10. The ‘#’ and ‘b’ represents sharp and flat in the musical sense respectively. Each pitch constant has an integer representation from 0 to 127 (see Appendix 1). This allows pitches to be incremented numerically and manipulated mathematically. The MIDI file format represents pitches in this manner, as well. Examples: C0, C#1, D9, etc. On declaration, pitches initialize to a default to a value of C0.

3.2.6. Types

Sounds
A sound is a special data structure containing a list of pitches, a double, and an integer separated by colons. The list of pitch literals can be thought of as a chord with attached duration and amplitude information. The double representation of quarter and half note durations would be, respectively, 0.25 and 0.5. The amplitude is represented as an integer ranging from 0 to 100. Example: |C4, E4, G4|:1/4:75. The fields in a sound can be set by calling the appropriate built-in function: setPitches, setDuration, or setAmplitude. Likewise, the values contained in each sound field can be retrieved by calling the appropriate built-in function: getPitches, getDuration, or getAmplitude. On declaration, sounds initialize to a default to a tuple of |C0|:0.0:0.

3.3. Dynamic Arrays

An array is a data structure that stores one or more elements of the same type. If an array has *n* elements, the array has length *n* and the components are referenced using indices from *0* to *n - 1*. An array's length is not fixed and can be increased by setting an element's value at an index greater than or equal to the current length.

3.3.1. Array Types

An array's type determines the category of object held in each of its cells. Types are determined when arrays are declared and cannot be subsequently changed. Extending an array results in the intermediate values being set to the default value for the array type.

3.3.2. Declaring Arrays

An array is declared by specifying the data type of its elements next to a set of brackets, followed by the array's name. At declaration time, arrays automatically initialize a single default value, based on their type, and have a length of 1.

```
int[] my_ints;                                    // my_ints = [0]
```

3.3.3. Assignment

After declaration, all the elements of an array can be assigned at once by listing the elements of the same type, separated by commas, inside a set of brackets.

```
my_array = [ 0, 1, 2, 3, 4 ];
```

3.3.4. Accessing and Manipulating Array Elements

An array element can be accessed by specifying a declared array's name, followed by the element's index, enclosed in a set of brackets. Values can be only be retrieved from the current range of indexes. However, values can be set at any index greater than zero. Setting an index beyond the current array range will fill the intermediate values with default values based on the arrays type.

```
int my_int;
my_int = my_array[3];                          // my_int = 3
my_array[9] = 10;                              // my_array = [ 0, 1, 2, 3, 4
                                                              0, 0, 0, 0, 10 ]
```

3.3.5. Other Array Operations

An array's length can be returned by calling the built-in length() function.

```
int length;
length = length(my_array); // length = 10;
```

3.4. Expressions

An expression in Lullabyte is a combination of explicit values, constants, variables, operators, and functions that are interpreted according to the following rules of precedence and association. The expression is evaluated and returns a value. The complete list of expressions below is grouped according to precedence. Each major subsection shares the same precedence, while the entire list is ordered from highest to lowest precedence. The associativity within the major subsections is stated below. For operators with two expressions the type rules are commutative. That is, if a rule is stated for the types <type_x> <operator> <type_y>, the rule also applies for <type_y> <operator> <type_x>.

3.4.1. Primary Expressions

Primary expressions are the most basic expression which allows for the construction of more complex expressions. Primary expressions are evaluated from left to right.

Identifiers
Identifiers' types are specified by their variable declarations including int, pitch, double, boolean, sound, and [] as arrays. Identifiers can be used for integers, doubles, booleans, functions, arrays, strings, pitches, and sounds.

Constants
Constants are integer constants, double constants, boolean constants and pitch constants.

(expression)
Expressions with parentheses are used to specify precedence.

3.4.2. Unary Operators

Unary operators invert or negate an expression. Unary operators are evaluated from left to right.

–expression
The operator – is the numerical negation. The negation operator, applied to an int, returns the negative int representation and, applied to a double, returns the negative of that double. Negation is not a valid operator for other types.

!expression
The operator ! is the logical not operator. If the expression is a boolean with value false, not applied to it returns true; if the boolean is true, not applied to it will return false. Booleans are the only type allowed.

3.4.3. Multiplicative Operators

Multiplicative operators are the expressions that perform multiplication and division. Multiplicative operations are evaluated from left to right.

*expression * expression*
The operator * is multiplication. An int * int will return an int. A double * double will return a double. An int * pitch and pitch * int will return a pitch at the corresponding pitch level. For instance, E1 converts to the value 16, so E1 * 2 will return G#2 which converts to 32 (see Appendix 1 for pitch conversions). A sound * int or int * sound will return a sound with the duration of the original sound multiplied by the amount of the int times. A sound * double or double * sound will return a sound with the duration of the original sound multiplied by the amount of the double times. An int * double or double * int will return an double of the expected value. An array * int or int * array will return a new array with the elements of the array repeated amount of the int times. No other combinations are allowed.

expression / expression
The / operator is division. The type considerations are the same as that of multiplication except with the operations that deal with sounds and arrays. Additionally, for a pitch / int, if the value is outside of the valid pitch range, [0 – 127], a compile time error will be thrown. An error will be thrown at compile time when an expression is divided by zero.

expression % expression
The % is the modulus operator. Expression % expression will return the remainder from the first expression divided by the second expression. The only modulus operations allowed are int % int, int % pitch, and pitch % int.

3.4.4. Additive Operators

Additive operators include addition and subtraction. Additive operators are evaluated from left to right.

expression + expression
The + operator is addition. If both expressions are integers, an integer is returned. An int + pitch or pitch + double will return the pitch at the corresponding value, i.e., adding 12 + C4 will return C5. This type is valid within range of the integer value of the pitch. A double + double and a double + int will return a double. These are the only types allowed.

expression – expression
The – operator indicates subtraction. The same type rules as addition apply here.

3.4.5. Relational Operators

Relational operators compare two expression's values. The <, >, <=, >= operators return the boolean false if the mathematical relation is false and return the boolean true if the relation is true. Valid relations include int to int, int to pitch and pitch to int, int to double and double to pitch, pitch to pitch, and double to double. There are no other type cases allowed. These expressions are evaluated from left to right.

expression < expression
expression > expression
expression <= expression
expression >= expression

3.4.6. Equality

The == and != operator evaluate whether two expressions are equivalent or not, respectively. The equality operators return the corresponding boolean values to the evaluated expression. Valid relations are the same as the relational operators, and also booleans. Equality operators are evaluated from left to right.

expression == expression
expression != expression

3.4.7. Logical And

The && is the logical AND operator. If two booleans are true, a true is returned otherwise when comparing two booleans a false boolean is returned. Only booleans are allowed to be compared. And expressions are evaluated from left to right.

expression && expression

3.4.8. Logical Or

The || operator indicates the logical OR operator. When comparing two booleans if either of the booleans is true or if both are true, then a true is returned. If both of the booleans are false then a false is returned. The only type case allowed is with booleans. The OR operator is evaluated from left to right.

expression || *expression*

3.4.9. Assignment

The = operator sets a variable to a specific evaluated expression. This is the only expression case that is right associative. The value and the evaluated expression must both be of the same type with no exceptions. Within a given scope, all variables are required to be declared before any expression is assigned a value.

expression = *expression*

3.5. Functions

Functions encapsulate a task by combining many instructions into a single line of code. Lullabyte provides several built-in functions to facilitate music composition. Additionally, developers can define their own functions.

3.5.1. Function Definitions

Function definitions start with the function's return type. If no value is to be returned, the return type void is to be used. The return type is followed by the function's name. The function's name is followed by a pair of parentheses that contain the function's parameters, if any, separated by commas when there are more than one. The parameters consist of the parameter's type and name separated by a space. Finally, the function's statements that the function executes are defined in between the first open brace and the corresponding closing brace as in C and C-derived languages. The type of the return value needs to match the return type. Here is an example of a function definition in Lullabyte:

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

3.5.2. Function Calls

Function calls can take place anywhere in a program where a variable of the same type as the function's return type is allowed. For example, the following code sets the value of a variable to the value returned by a function:

```
/* Using a function to define a variable. */
sound wagonWheelQuarters[] = [
|G4, B5, D5, G5|:1/4:50,
|D5, F#5, A5, D4|:1/4:50,
|E4, G4, B5, E5|:1/4:50,
|C5, E5, G5, C6|:1/4:50];
sound oneChordPerMeasure[]= quadruple(wagonWheelQuarters);
```

Functions can also execute code without returning a value. In these cases, the default values are returned when evaluated. The default value is assigned on function declaration and is analogous to type declarations of variables. Void functions are assigned a value of Integer 0.

3.5.3. Built-In Functions

Lullabyte utilizes very few built-in functions. Most of the functionality that the user needs can be created by writing custom functions, but a few built-in functions are required to generate MIDI output files, as well as interact with the fields of sound objects.

void bpm(int beats_per_minute)
The bpm function accepts an int as an argument and sets the beats per minute on the resulting output file. MIDI bases bpm on pulses per clock. For most processors we suggest a value between 40 and 300. Values outside this range will be accepted, but their execution is undefined. Calling mixDown before bmp will default every mixDown call's bpm thereafter to a value to 120.

void mixDown(sound[] to_append, int track_num)
The mixDown function accepts an array of sounds or a single sound and an optional int as its arguments. The sound or sound array is the music intended on being written and/or played to the midi file and the int is the music's corresponding track. The default track, if no int is specified, is 0. Each mixDown call appends the music to the specified track in the bytecode. If bpm hasn't yet been called, each mixDown call will set the music to a beats per minute of 120. If play or write (explained below) is not specified, the midi music will be played and written to a file.

play()
The play function sets the midi environment to only play the music that is passed to mixDown and is called with no arguments. The play function must be called before mixDown to take effect. The play function should not be called in the same program with the write function called. In that case, the more recent of the two called will take effect.

write()
The write function sets the midi environment to only write the music that is passed to mixDown to a midi file and is called with no arguments. The write function must be called before mixDown to take effect. The play function should not be called in the same program with the write function called. In that case, the more recent of the two called before will take effect.

pitch[] getPitches(sound s)
The getPitches() function accepts a sound object as its argument and returns an array of the pitches contained in the that sound's pitches field.

double getDuration(sound s)
The getDuration() function accepts a sound object as its argument and returns the double contained in that sound's duration field.

int getAmplitude(sound s)
The getAmplitude() function accepts a sound object as its argument and returns the int contained in that sound's amplitude field.

sound setPitches(sound s, pitch[] p)
The setPitches function accepts a sound and an array of pitches as arguments and returns a new sound with the pitches set to the input argument. Other attributes of the new sound are equal to the original.

sound setDuration(sound s, double d)
The setDuration function accepts a sound and double as arguments and returns a new sound with the duration set to the input argument. Other attributes of the new sound are equal to the original.

sound setAmplitude(sound s, int i)
The setAmplitude function accepts a sound and an integer as arguments are returns a new sound with the amplitude set to the input argument. Other attributes of the new sound are equal to the original.

int length(data_type array[])
The length function accepts an array object as its argument and returns the number of elements in that array as an int.

int randomInt(int bound)
The randomInt function accepts an integer as its argument and returns a random integer between 0 and the bound, exclusively.

double randomDouble(double bound)
The randomDouble function returns a random number between 0 and the bound, exclusive.

3.6. Statements

A statement is the smallest standalone element of Lullabyte. A program written in Lullabyte is formed by a sequence of one or more statements. Statements are executed in sequential order. They are used for assignment, function calls, and control flow.

3.6.1. Expression Statements

Expression statements have the form:

```
expression;
type variable;
```

They are the most commonly used statements in Lullabyte. Expression statements are used for variable declaration, function calls, and checking of conditions. At declaration, an expression takes on a default value based on its data type. Ints default to 0; doubles default to 0.0; booleans default to false; pitches default to C0; and sounds default to |C0|:0.0:0. After the assignment, the expression on the left side of the statement takes on the provided value.

3.6.3. Conditional Statements

Conditional statements have the form:

```
if ( expression ) {
      // statement-list-A;
}
else {
      // statement-list-B;
}
```

The if statement evaluates an expression and checks whether the expression is true. If it is true, then statement-list-A is executed. Otherwise, statement-list-B is executed. The else statement is optional. Both statements require braces around their respective list of statements.

3.6.4. For Statement

For statements have the form:

```
for (initialization; termination; increment) {
      // statement-list
}
```

The initialization expression executes once, as the loop begins. The termination expression is evaluated at the beginning of each loop; the loop exits when termination evaluates to false. The increment expression is invoked after each iteration. Typically, a variable is assigned a value in the initialization expression, incremented or decremented in the increment expression, and compared to a terminating value in the termination expression.

3.6.5. While Statement

The while statement has the form:

```
while ( expression ) {
     // statement-list;
}
```

The while statement evaluates an expression and checks whether or not the expression is true. If it is true, then the program enters a loop: the statement-list is executed; the expression is re-evaluated. The loop continues as long as the expression is evaluated to be true, exiting when the expression is evaluated to be false.

3.6.6. Loop Statement

The loop statement has the form:

```
loop ( <var> : <array> ) {
     statement-list;
}
```

The loop statement takes a variable and an array and iterates through every single element in the array. On each iteration, <var> becomes the ith element of the array and it behaves as if it were "array[i]." If <var> is being assigned to, the value of the ith element of the array will be replaced. Otherwise, the value of ith array element will be returned. The variable's type must match the type of the elements in the array and it must be declared in earlier in the code as any other variable is declared.

3.6.7. Return Statement

The return statement has the form:

```
return expression;
```

Return statements are used to terminate a function and return a value to its caller. In the first case, no value is returned and is only valid for functions with the return type void. In the second case, the value of the expression is returned to the calling function and is only valid for functions with a non-void return type.

3.7. Scope Rules

The scope of an identifier in a Lullabyte program is the section of the program where the identifier may be accessed.

3.7.1. Global Scope

Entities with global scope are declared outside of function definitions. They can be used anywhere in the Lullabyte program. Globally scoped identifiers cannot be overwritten by locally scoped identifiers.

```
/*
 * chord_x can be accessed by foo() because it has global scope
 */
sound[] snds_x;

function sound[] foo() {
   sound snds_y[] = snds_x*4;
   return snds_y;
}

void main(){
      snds_y;
snds_x = [[A5, D5, G5]:1/2:50, [A5, D5]:1/2:50] ;
      snds_y = foo();
}
```

3.7.2. Local Scope

Entities with local scope are declared inside a set of braces. Entities with local scope can only be used inside the scope of those braces.

```
/*
 * chord_y's scope is restricted within the context of foo(). It cannot be accessed anywhere
 * else such as foo2(). Similarly, the scope of factor is restricted within the if block in
 * foo().
 */
sound[] foo(boolean extend) {
      sound[] snds_y;
      int factor;
      snds_y = [A5, D5, G5];
      if(extend) {
              factor = 4;
              snds_y = snds_y * factor;
      }
      return snds_y;
}

sound[] foo2() {
      sound[] snds_x;
      snds_x = [G5, E5, C5];
      return snds_x*4;
}
```

3.8. File Format and Output

Generating a MIDI file from Lullabyte involves transforming a collection of Sounds into the JFugue MusicString format in a Java file and compiling and running the resulting Java code.

Collection of Sounds in Lullabyte → Java code using JFugue library → MIDI

The Lullabyte compiler transforms its collection of Sounds into Java Strings, following the JFugue MusicString format. The C major chord, for example, would become:

```
“V0 [72]/0.25a100+[76]/0.25a100+[79]/0.25a100”
```

The V0 indicates this is track 0, the [72] is the pitch according to the midi pitch map, the 0.25 is the duration, and the a100 is the amplitude. The + indicates these notes will be played simultaneously.

In Lullabyte, multiple arrays of notes can be played in parallel by passing the arrays to separate mixDown function calls. To specify the track, mixDown should be called with two arguments, the sequence of sounds, and the track. mixDown being called with different tracks will be played simultaneously, and mixDown calls with the same track will be appended together. Each array of sounds passed to the mixDown function is transformed into a Java String and added as a JFugue Pattern through the translated bytecode. The transformation of parallel sounds is shown below for V-IV-I chord progression in C and the corresponding base notes:

```
sound[] chords;
sound[] bass;


chords = [|G5, B5, D5|:0.25:100, |F5, A5, C5|:0.25:100, |C5, E5, G5|:0.5:100];
bass = [|G3|:0.25:100, |F3|:0.25:100, |C3|:0.25:100];
mixDown(chords, 0);
mixDown(bass, 1);
```

This lullabyte code will generate the following bytecode.

```
120 b
0[[D5, B5, G5]:0.25:100,[C5, A5, F5]:0.25:100,[G5, E5, C5]:0.5:100]
1[[G3]:0.25:100,[F3]:0.25:100,[C3]:0.25:100]
```

Here, the 120 at the top indicates the beats per minute. To change the bpm of the midi file, bpm(int) must be called before mixDown(). The “b” in the first line of the bytecode indicates that the BytecodeTranslator.java will both write and play the midi file. If play() or write() was called prior to mixDown() the indicator will be p or m respectively and the midi file will only be played or written. bpm(), play(), and write() must be called before mixDown() to take effect. If the compiler fails in any way the bytecode will be written to a single “x” char. This will indicate to BytecodeTranslator there was a failure and the midi file should not be played or written. From the generated bytecode, the BytecodeTranslator generates one Java String containing the proper format for JFugue:

```
...
Pattern p1 = new Pattern();
p1.add(“V0 [67]/0.25a100+[71]/0.25a100+[62]/0.25a100 ”+
“[65]/0.25a100+[69]/0.25a100+[60]/0.25a100 ” +
“[60]/0.5a100+[64]/0.5a100+[67]/0.5a100”);
p1.add(“V1 [43]/0.25a100 [41]/0.25a100 [36]/0.5a100”);
try {
       player.saveMidi(p1, new File(title + ".mid"));
} catch (IOException e){
       System.out.println(e);
}
...
```

This, in turn, generates a midi file with the chords and bass specified above, played in parallel. The midi file is defaulted to llb-write.mid, but can be specified by the second argument of ./lullabyte.

4. Project Plan

4.1. Planning

The Lullabyte team had regular team meetings every Sunday night at 8:30pm to give updates on progress and plan out subsequent steps of development. Team members who did not have class conflicts met regularly with our project advisor, Julian Rosenblum, on Tuesdays at 4:00pm to discuss current issues and receive advice.

4.2. Development

The Lullabyte project employed an iterative, distributed development model as suggested by the git version control system. Production code was stored on a master branch that was saved on the internet and that all team members had access to. Team members were then assigned specific tasks that they pursued individually on local branches. For each task, an independent test file was created to test functionality. For a task to be considered complete, the new code would have to pass its own test file, as well as all previous test files. When tasks were completed, the local branch was merged with the master branch and pushed to all team members' code.

4.3. Responsibilities

Every team member worked on the parser, AST, and the interpreter, ensuring that any team member could work autonomously to add features to the compiler. Each member was responsible for a section of the interpreter, and was required to add to the parser and AST if that was needed for their section. In the backend, Nathan Hayes-Roth was responsible for statements, Ben Nappier focused on the mixDown functionality and the bytecode generation, Stanley Chang concentrated on types and type consistency, Louis Croce focused on operators, Andrew Langdon implemented the built-in functions, and Peter Xu focused on control flow, functions and arrays. Each team member wrote tests for their section when necessary.

4.5. Timeline

The Lullabyte team followed the timeline outline for the class project, only adding two additional milestones: a Hello World program that would print all of our object types and a functional mixDown function that would build a MIDI file.

Milestone	Date
Proposal	September 25
Language Reference Manual	October 28
Hello World	November 11
mixDown	December 1
Final Report	December 20

4.6. Log

See Appendix 3 for a complete listing of our group’s git commits.

4.7. Style Guide

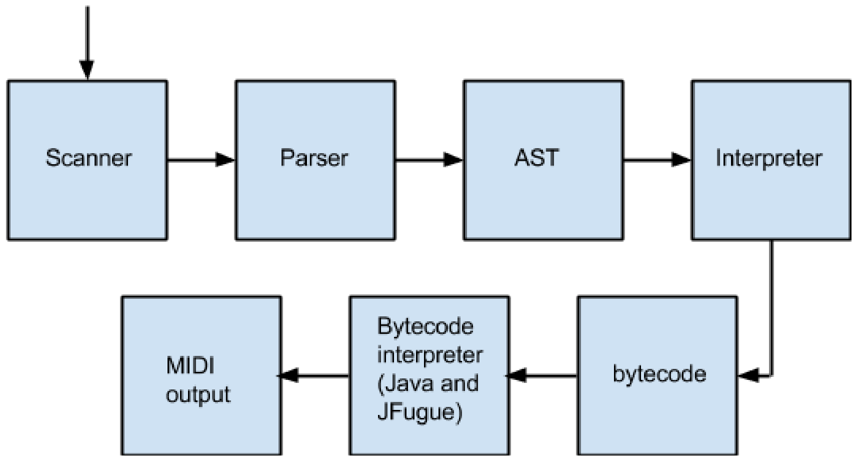
The Lullabyte team had an informal approach to programming style. We attempted to follow the coding styles presented on the ocaml.org website and in the MicroC compiler examined during the course of the term. We were not concerned with legibility when printed, only that it was readable amongst team members. Beyond that, our essential style guidelines were as follows:

- Code must compile
- All tests must pass
- Indent using tab characters
- Use meaningful names
- Comments go above the code they reference
- Avoid useless comments
- Avoid over-commenting

4.8. Development Environment

The Lullabyte compiler was primarily written in Objective Caml (OCaml), with elements of Java, Ruby, and shell scripting . The bulk of the compilation is done in OCaml, which generates bytecode that is run through a Java program to generate the resulting MIDI files. The test script is written in Ruby. Team members developed code on Ubuntu and Mac OS X machines using Sublime Text 2, Eclipse, and each operating system’s respective terminal.

5. Architectural Design



Our compiler takes in a stream of bytes from the .llb file and tokenizes them using scanner.ml. Then, these tokens are passed to the parser, which extracts information from the token and enforces the syntax of our language. Then, the program is structured as an abstract syntax tree, which is forwarded to our interpreter. The interpreter recursively walks the tree, and produces a file of bytecode. This bytecode is read by the bytecode interpreter which is written in Java and uses the JFugue library. The bytecode interpreter produces the MIDI output.

The primary contributor to the scanner was Ben Nappier. The parser was mostly written by Peter Xu and Andrew Langdon but each team-member contributed to it when they needed to modify the parser for the functionality they were responsible for. The AST was mostly written by Peter Xu and Andrew Langdon. The interpreter was a collaborative effort (see 4.3. Responsibilities). The Bytecode generator was written by Ben Nappier.

5.1. Frontend

5.1.1. Scanner

The scanner recognizes all of our data types, operators, and delimiters, including sounds and pitches. It also removes commented code.

5.1.2. Parser

The parser verifies the syntax of a lullabyte file and also builds up the AST. It binds values to OCAML types for later use by the interpreter.

5.1.3. Abstract Syntax Tree

The abstract syntax tree contains the contents of the lullabyte file, ready to be walked by the interpreter. It defines the structure of the tree and how the data is stored. Our ast.ml also contains helper functions for recursively printing a program tree.

5.2. Backend

5.2.1. Interpreter

The interpreter walks the ast and, after executing the instructions contained in the tree, writes a file of bytecode that contains the pitch, duration, and amplitude data about the final output midi notes.

4.2.2. Bytecode

The bytecode contains a line of notes in a similar format to the representation in lullabyte files. Each line represents a track, which are played/written simultaneously. The first line contains flags and the BPM of the output file.

5.2.3. Bytecode Interpreter

The bytecode interpreter is a Java program that reads in the bytecode line-by-line. It takes the note specifications for each track and then uses JFugue library functions to write these notes to midi. A JFugue library function writes the file to midi.

6. Test Plan

6.1. Overview

With six contributors to this project, we needed a test suite to feel confident that changes made in one section of our interpreter were not breaking things in other parts of the compiler. We split the tests into two sections, frontend tests and backend tests. The frontend tests make sure that an .llb file can be correctly parsed into an AST. The backend tests ensure that our code behaves as expected. A test script compiles our source code and then runs all the tests against it, displaying errors if there are any and displaying pass messages if the specific test works.

The frontend tests are fairly simple. Each .llb file in the test folder is run through our scanner and parser. If the file contains a syntax error, an error is thrown and the test fails.

Each backend test consists of two files, one expected output file and one .llb file. Our test script finds each .out output file and runs the corresponding .llb file. If the output from running the .llb program matches the expected output in the .out file, the test passes. To add a new test, you just create the .out file and the .llb file, and the script will find the .out file and run the test. There is no need to modify the script itself to add new tests.

6.2. Test Script

The test script is written in Ruby:

```
#!/usr/bin/env ruby

argsString = ARGV.join.gsub('-', '')


if defined?(argsString) && argsString.include?("d")
	dump_flag = true
else
	dump_flag = false
end


if defined?(argsString) && argsString.include?("w")
	supress_string = " > /dev/null 2>&1"
else
	supress_string = ""
end

if defined?(argsString) && argsString.include?("o")
	show_output = true
else
	show_output = false
end

(continued on next page)
```

(continued from previous page)

```
# remove junk and compile
`rm interpret > /dev/null 2>&1`
`rm printAst > /dev/null 2>&1`
`rm *.cmo > /dev/null 2>&1`
`rm *.cmi > /dev/null 2>&1`
`ocamllex scanner.mll #{supress_string}`
`ocamlyacc parser.mly #{supress_string}`
`ocamlc -i ast.ml > ast.mli`
`ocamlc -c ast.mli #{supress_string}`
`ocamlc -c ast.ml #{supress_string}`
`ocamlc -c parser.mli #{supress_string}`
`ocamlc -c scanner.ml #{supress_string}`
`ocamlc -c parser.ml #{supress_string}`
`ocamlc -c printAst.ml #{supress_string}`
`ocamlc -c helper.ml #{supress_string}`
`ocamlc -c interpreter.ml #{supress_string}`
`ocamlc -o interpret ast.cmo helper.cmo parser.cmo scanner.cmo interpreter.cmo
#{supress_string}`
`ocamlc -o printAst ast.cmo parser.cmo scanner.cmo printAst.cmo #{supress_string}`

#
# Frontend tests!
#

puts "\n\n\e[47m\e[30mFrontend Tests:\e[0m"
# puts "\n\nFrontend Tests:"
all_pass = true
Dir['tests/*.llb'].each do |filePath|
	output = `./printAst < "#{filePath}"`


	if output == "syntax error\n"
		puts "\e[31m#{filePath}"
		all_pass = false
	end
end

if all_pass == true
	puts "\e[32mAll frontend test files were correctly parsed."
end
print "\e[0m"
```

(continued on next page)

```
(continued from previous page)

#
# Backend tests!
# For each .out file in the tests directory,
# find the corresponding .llb file,
# run it,
# and compare the .out file to the actual output
#

puts "\n\n\e[47m\e[30mBackend Tests:\e[39m\e[0m"
# puts "\n\nBackend Tests:"
Dir['tests/*.out'].each do |filePath|
      expected_output = File.read(filePath)
      test_file = filePath.gsub('.out', '.llb')

      print "\e[0m"

      actual_output = `./interpret < #{test_file}`
      test_name = test_file.gsub('tests/', '')
                                          .gsub('.llb', '')

      if expected_output == actual_output
              puts "\e[32m" + test_name + " passed!\e[0m"
      elsif expected_output == actual_output && show_output
              puts "\e[32m" + test_name + " passed!\e[0m"
              puts expected_output if dump_flag == false
              puts expected_output.dump if dump_flag == true
      else
              puts "\e[31m" + test_name + " failed. \nIt could be a whitespace issue! Run with
-d to see whitespace"
              puts "\e[37m---Expected_output: \n" + expected_output if dump_flag == false
              puts "\e[31m---Actual output: \n" + actual_output if dump_flag == false
              puts "\e[37m---Expected_output: \n" + expected_output.dump if dump_flag == true
              puts "\e[31m---Actual output: \n" + actual_output.dump if dump_flag == true
      end
end

puts "\e[0m"
```

6.3. Test Suite

Team members were responsible for writing tests for their contributions before pushing code. The complete set of backend tests has been included in Appendix – 2. As an example, here's a test and the related output file for 'for' loops:

test-for.llb

```
/*
 * Testing FOR statements.
 */


void main(){


   int i;
   int j;


   /*
    * Basic for loop
    */
   for (i = 0 ; i < 5 ; i = i + 1) {
       print(i);
   }
   print(42);


   /*
    * For within a for.
    */
   for (i = 0; i< 5; i=i+1) {
     for (j=5-i; j>0; j=j-1){
          print(i + 1);
     }
   }
}
```

test-for.out

```
0
1
2
3
4
42
1
1
1
1
1
2
2
2
2
3
3
3
4
4
5
```

7. Lessons Learned

7.1. Stanley Chang

Type checking should be planned much earlier in the developing stage. Unlike testing for the right execution of the program, type checking requires to check all the ways the program should not compile. Since type checking can't be done until most of the compiler is completed, theorizing which tokens need to be saved and accessed will allow a smoother transition to type checking. I started type checking realizing types of variables and functions weren't being saved and needed to restructure the way we type check by using the initialized value. The initial patch can spawn more bugs that require more patching. Overall, type checking is very enjoyable because it allowed me to have a deeper understanding of the compiler.

7.2. Louis Croce

Working in a large group can be a challenging, but when done right, can be very advantageous. There were many reasons why our group was able to succeed in a large group, but I think most stem from us knowing each other pretty well from the start and most of us living together. Accountability and communication are two keys to any successful development team. By knowing each other well from the onset, accountability was not an issue. We were comfortable enough with each other to call one another out when someone wasn't getting his current job done. At the same time, if one of us was having a rough week, it was easy to communicate why something wasn't getting done and when it will be completed. Strong communication was consistent in our group and was easy to come by with our weekly (and many times semi-weekly) team meetings.

7.3. Nathan Hayes-Roth

Most importantly: version control. If any of your team members don't know how to use git, teach them immediately. Collaboratively writing a code base with 3-6 people would be virtually impossible to manage, otherwise. Secondly, meet with your group and your advisor frequently and keep logs of all meetings and conversations. You might think you're meeting too frequently, or that you have nothing to talk about. But chances are, someone in the group has hit a roadblock or is considering a problem that the other members either know the answer to, or haven't considered before. Finally, be decisive. Meeting with your group frequently is important, but if you spend the whole time debating the merit of some feature, you won't be able to move forward until the next meeting. The semester is too short to waste time debating a feature that might never make it into production. Make decisions and stick to them. If you have time at the end, you can reconsider certain features that you passed on earlier.

7.4. Andrew Langdon

Get hello world working as soon as possible. Once you understand how the logic works from the bottom up it's easy to add code to the compiler. Also, get a test suite up right away so you can be sure you're not breaking anything. Don't be afraid to completely delete a block of code or a method and start over. You'll probably write it better the next time around. Don't spend too long talking. Discuss the best options, decide on one, and start to code. You’ll quickly realize if the other ways are better.

7.5. Ben Nappier

At first, this project seemed impossible. Without having hello world in early November, I thought we might not finish. Things turned around, and the project ended up being a fun and valuable experience. Documenting any and every language design change greatly increased the productivity of our meetings. Before doing this, we spent too much time debating the same topics. Having everyone attempt hello world forced the whole group to have a basic understanding of the entire compiler flow. This was helpful in that group members could assist one and other in sections that were not their primary responsibility or specialty. Helping each other was the key to efficiency. Pair programming with our teammates increased our code’s readability and minimized the time spent banging our heads with OCaml errors. Throughout development of our language, I felt we spent too much time debating the design. However, in the end, I felt the extra time spent debating was well worth it.

7.6. Peter Xu

Building your compiler/interpreter starting from the ground up is a great way to learn and get familiarized with the structure of the compiler, scanner, parser, ast, etc. and how data is passed around. It is helpful to reference MicroC while doing this and incrementally add more functionality to the compiler. As the team leader, some important lessons I learned for managing the project were to keep a schedule of weekly milestones, sharing progress made with your teammates, and taking the initiative to implement. If the group has not been making much progress code-wise, it is good if the leader can take initiative and get the ball rolling.

Appendix 1 - Midi Pitch Map

Midi pitch map:

Octave	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	0	1	2	3	4	5	6	7	8	9	10	11
1	12	13	14	15	16	17	18	19	20	21	22	23
2	24	25	26	27	28	29	30	31	32	33	34	35
3	36	37	38	39	40	41	42	43	44	45	46	47
4	48	49	50	51	52	53	54	55	56	57	58	59
5	60	61	62	63	64	65	66	67	68	69	70	71
6	72	73	74	75	76	77	78	79	80	81	82	83
7	84	85	86	87	88	89	90	91	92	93	94	95
8	96	97	98	99	100	101	102	103	104	105	106	107
9	108	109	110	111	112	113	114	115	116	117	118	119

Appendix 2 - Complete Code Listing

BytecodeTranslator.java

```
import java.io.*;
import org.jfugue.*;

public class BytecodeTranslator {
       /***
          To compile: javac -classpath ./jfugue-4.0.3.jar BytecodeTranslator.java
          To run: java -cp jfugue-4.0.3.jar:. BytecodeTranslator [filename]
    ***/
       public static void main(String[] args) {
              if (args.length != 2){
                     System.out.println("must give a bytecode filename and a midifile name");
                     System.out.println(args.length);
                     System.exit(1);
              }
              String byteCodeFileName = args[0];
              String midiFileName = args[1];
              BufferedReader br = null;
              Player player = new Player();
              Pattern p = new Pattern();
              String currentLine;
              String byteCode = "";
              String[] tracks = new String[16];
              String[] mixDownWrites;
              String[] firstLine;
              String option = "";

              try {

                     String tempo = "T";
                     int first = 1;
                     br = new BufferedReader(new FileReader(byteCodeFileName));
                     while ((currentLine = br.readLine()) != null) {
                            if(first ==1){
                                   if(currentLine.charAt(0) == 'x'){
                                          // System.err.println("Lullabyte file failed or
                                          // mixdown was not called.");
                                          System.exit(0);
                                   }
                                   firstLine = currentLine.split(" ");
                                   tempo = tempo + firstLine[0] + " ";
                                   option = firstLine[1];
                            } else {
                                   byteCode += currentLine.charAt(0) + currentLine
                                   .substring(2, currentLine.length()-1) + "\n";
                            }
                            first --;
                     }

                     mixDownWrites = byteCode.split("\n");

                     for(int i = 0; i<mixDownWrites.length; i++){
                            int trackNum = Integer.parseInt(
                                   String.valueOf(mixDownWrites[i].charAt(0)));
                            if(tracks[trackNum] == null){
                                   tracks[trackNum] = mixDownWrites[i].substring(1);
```

```
        } else{
                tracks[trackNum] += mixDownWrites[i].substring(1);
        }
}

String midiWrite = "";

for (int i=0;i<tracks.length;i++){
        if(tracks[i] != null){
                String track="";
                String chord="";
                if(i > 0){
                        track += " " + "V" + i + " ";
                } else {
                        track += "V" + i + " ";
                }

                String[] sounds = tracks[i].split("\\[");

                for (int j=1;j<sounds.length;j++){
                        String[] chordsDurAmp = sounds[j].split(":");

                        String amp = "";
                        if(chordsDurAmp[2].endsWith(",")){
                                amp = chordsDurAmp[2].replace(",", "");
                        }else{
                                amp = chordsDurAmp[2];
                        }
                        double durr;
                        if (chordsDurAmp[1].indexOf(".") == -1){
                                String[] rat = chordsDurAmp[1].split("/");
                        durr = Double.parseDouble(rat[0]) /
                                Double.parseDouble(rat[1]);
                        } else {
                                durr = Double.parseDouble(chordsDurAmp[1]);
                        }

                        String[] chords = chordsDurAmp[0].split(" ");
                        for(int l=0;l<chords.length-1;l++){
                                chord += chords[l].substring(
                                        0, chords[l].length()-1) +
                                        "/" + durr + "a" + amp + "+";
                        }
                                chord += chords[chords
                                        .length-1].substring(
                                        0, chords[chords.length-1]
                                        .length()-1) + "/" + durr + "a" +
                                        amp + " ";
                }
                try {
                        chord = chord.substring(0, chord.length()-1);
                } catch (StringIndexOutOfBoundsException e){
                        System.out.println(
                                "Mixdown must be called on an array of
                                sounds");
                        System.exit(1);
                }
                track += chord;
                midiWrite += track;
        }
}
```

```
            midiWrite = tempo + midiWrite;
            p.add(midiWrite);
            if(option.equals("w")){
                System.out.println("Writing tracks to midi file");
                player.saveMidi(p, new File(midiFileName));
            }
            if(option.equals("p")){
                System.out.println("Playing llb tracks");
                player.play(p);
            }
            if(option.equals("b")){
                System.out.println("Writing and playing midi file");
                player.saveMidi(p, new File(midiFileName));
                player.play(p);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (br != null)br.close();
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

Makefile

```
all:
	javac -classpath ./jfugue-4.0.3.jar BytecodeTranslator.java
	ocamllex scanner.mll;
	ocamlyacc parser.mly;
	ocamlc -i ast.ml > ast.mli;
	ocamlc -c ast.mli;
	ocamlc -c ast.ml;
	ocamlc -c parser.mli;
	ocamlc -c scanner.ml;
	ocamlc -c parser.ml;
	ocamlc -c helper.ml;
	ocamlc -c interpreter.ml;
	ocamlc -o interpret helper.cmo ast.cmo parser.cmo scanner.cmo interpreter.cmo;


clean:
	rm -rf *.cmo
	rm -rf *.cmi
	rm -rf *.mli
	rm -rf BytecodeTranslator.class
	rm -rf interpret
	rm -rf bytecode
	rm -rf *.mid
```

ast.ml

```
type op = Add | Sub | Mult | Div | Mod | Lt | Gt | Leq | Geq | Eq | Neq | And | Or
type typeConst = Integer | Double | Void | Pitch | Sound | Boolean

type expr =
         Int of int
       | Double of float
       | Boolean of bool
       | Pitch of string
       | Sound of string list * float * int
       | Id of string
       | Array of expr list
       | Index of string * expr list
       | Call of string * expr list
       | Assign of expr * expr
       | Binop of expr * op * expr
       | Not of expr
       | Neg of expr

type stmt =
         Block of stmt list
       | Expr of expr
       | Return of expr
       | If of expr * stmt * stmt
       | For of expr * expr * expr * stmt
       | While of expr * stmt
       | Loop of string * string * stmt

type var_decl = {
       varname : string;
       vartype : string;
}

type par_decl = {
       paramname : string; (* Name of the variable *)
       paramtype : string; (* Name of variable type *)
}

type func_decl = {
       rtype : string;
       fname : string;
       formals : par_decl list;
       locals : var_decl list;
       body : stmt list;
}


type program = var_decl list * func_decl list


let rec string_of_expr = function
         Int(i) -> string_of_int i
       | Double(d) -> string_of_float d
       | Boolean(b) -> string_of_bool b
       | Pitch(p) -> p
       | Sound(p,d,a) -> "|" ^ String.concat ", " p ^ "|"
                             ^ ":" ^ string_of_float d
                             ^ ":" ^ string_of_int a
       | Id(s) -> s
       | Array(s) ->
              "[" ^ String.concat ", " (List.map string_of_expr s) ^ "]"
       | Index(s, i) -> s ^ "[" ^ String.concat "[" (List.map string_of_expr i) ^ "]"
       | Call(f, el) ->
              f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
       | Assign(id, stuff) ->
              (string_of_expr id) ^ " = " ^ (string_of_expr stuff)
       | Binop(e1, o, e2) ->
              string_of_expr e1 ^ " " ^
                     (match o with
                       Add         -> "+"
                     | Sub         -> "-"
                     | Mult        -> "*"
                     | Div         -> "/"
                     | Mod         -> "%"
                     | Or          -> "||"
                     | And         -> "&&"
                     | Eq          -> "=="
                     | Neq         -> "!="
                     | Lt          -> "<"
                     | Gt          -> ">"
                     | Leq         -> "<="
                     | Geq         -> ">="
                     ) ^ " " ^ string_of_expr e2
       | Not(e) -> "!" ^ (string_of_expr e)
       | Neg(e) -> "-" ^ (string_of_expr e)


let rec string_of_stmt = function
       Block(stmts) ->
              "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^ "}\n"
       | Expr(expr) -> string_of_expr expr ^ ";\n"
       | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n"
       | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^ string_of_stmt s
       | If(e, s1, s2) -> "if (" ^ string_of_expr e ^ ")\n" ^
              string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
       | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ; " ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
       | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^ string_of_stmt s
       | Loop(v, a, s) -> "loop (" ^ v ^ ":" ^ a ^ ") " ^ string_of_stmt s

let string_of_vdecl vdecl = vdecl.vartype ^ " " ^ vdecl.varname ^ ";\n"
let string_of_pdecl pdecl = pdecl.paramtype ^ " " ^ pdecl.paramname

let string_of_fdecl fdecl =
       fdecl.rtype ^ " " ^ fdecl.fname ^ "("
              ^ String.concat ", " (List.map string_of_pdecl fdecl.formals) ^ ")\n{\n" ^
       String.concat "" (List.map string_of_vdecl fdecl.locals) ^
       String.concat "" (List.map string_of_stmt fdecl.body) ^
       "}\n"

let string_of_program (vars, funcs) =
       String.concat "" (List.map string_of_vdecl (List.rev vars) ) ^ "\n" ^
       String.concat "\n" (List.map string_of_fdecl (List.rev funcs) ) ^ "\n"
```

helper.ml

```
module NameMap = Map.Make(struct
  type t = string
  let compare x y = Pervasives.compare x y
end)
module IntMap = Map.Make(struct
  type t = int
  let compare x y = Pervasives.compare x y
end)
let maxPitchInt = 119;;
let minPitchInt = 0;;
let pitchToIntMap = NameMap.empty;;
 (* let pitchToIntMap = NameMap.add "Cb" 0 pitchToIntMap;;   *)
let pitchToIntMap = NameMap.add "C" 0 pitchToIntMap;;
let pitchToIntMap = NameMap.add "C#" 1 pitchToIntMap;;
let pitchToIntMap = NameMap.add "Db" 1 pitchToIntMap;;
let pitchToIntMap = NameMap.add "D" 2 pitchToIntMap;;
let pitchToIntMap = NameMap.add "D#" 3 pitchToIntMap;;
let pitchToIntMap = NameMap.add "Eb" 3 pitchToIntMap;;
let pitchToIntMap = NameMap.add "E" 4 pitchToIntMap;;
let pitchToIntMap = NameMap.add "E#" 5 pitchToIntMap;;
let pitchToIntMap = NameMap.add "Fb" 4 pitchToIntMap;;
let pitchToIntMap = NameMap.add "F" 5 pitchToIntMap;;
let pitchToIntMap = NameMap.add "F#" 6 pitchToIntMap;;
let pitchToIntMap = NameMap.add "Gb" 6 pitchToIntMap;;
let pitchToIntMap = NameMap.add "G" 7 pitchToIntMap;;
let pitchToIntMap = NameMap.add "G#" 8 pitchToIntMap;;
let pitchToIntMap = NameMap.add "Ab" 8 pitchToIntMap;;
let pitchToIntMap = NameMap.add "A" 9 pitchToIntMap;;
let pitchToIntMap = NameMap.add "A#" 10 pitchToIntMap;;
let pitchToIntMap = NameMap.add "Bb" 10 pitchToIntMap;;
let pitchToIntMap = NameMap.add "B" 11 pitchToIntMap;;
 (* let pitchToIntMap = NameMap.add "B#" 12 pitchToIntMap;;  *)
let intToPitchMap = IntMap.empty;;
let intToPitchMap = IntMap.add 0 "C" intToPitchMap;;
let intToPitchMap = IntMap.add 1 "C#" intToPitchMap;;
let intToPitchMap = IntMap.add 2 "D" intToPitchMap;;
let intToPitchMap = IntMap.add 3 "Eb" intToPitchMap;;
let intToPitchMap = IntMap.add 4 "E" intToPitchMap;;
let intToPitchMap = IntMap.add 5 "F" intToPitchMap;;
let intToPitchMap = IntMap.add 6 "F#" intToPitchMap;;
let intToPitchMap = IntMap.add 7 "G" intToPitchMap;;
let intToPitchMap = IntMap.add 8 "G#" intToPitchMap;;
let intToPitchMap = IntMap.add 9 "A" intToPitchMap;;
let intToPitchMap = IntMap.add 10 "Bb" intToPitchMap;;
let intToPitchMap = IntMap.add 11 "B" intToPitchMap;;
let pitchToInt = fun x ->
       let octave = String.get x ((String.length x)-1) in
              let basicPitch = String.sub x 0 ((String.length x)-1) in
                     ((NameMap.find basicPitch pitchToIntMap) +
                     ((int_of_char octave) - 48) * 12)
(* There is nothing here to see if you are out of range *)

let intToPitch = fun x ->
       if x > maxPitchInt then raise (Failure ("Pitch higher than allowable threshold"))
       else if x < minPitchInt then raise (Failure ("Pitch lower than allowable threshold"))
       else (IntMap.find (x - 12*(x / 12)) intToPitchMap) ^ (string_of_int (x / 12))
```

interpreter.ml

```
open Ast open Printf open Helper

module NameMap = Map.Make(struct
  type t = string
  let compare x y = Pervasives.compare x y
end)

module TypeMap = Map.Make(struct
  type t = string
  let compare x y = Pervasives.compare x y
end)

let _ = Random.self_init()

(* Returns the type of an expression v *)
let getType v =
  match v with
    Int(v) -> "int"
    | Double(v) -> "double"
    | Boolean(v) -> "bool"
    | Pitch(v) -> "pitch"
    | Sound(p,d,a) -> "sound"
    | Array(v::_) -> "array"
    | _ -> "unmatched_type"

(* Returns the evaluation of the boolean expression v *)
let getBoolean v =
  match v with
    Boolean(a) -> a
    | _ -> false

exception ReturnException of expr * expr NameMap.t

(* Sets the default initialization value for a given type t *)
let initType t =
  match t with
    "int" -> Int(0)
    | "double" -> Double(0.0)
    | "bool" -> Boolean(false)
    | "pitch" -> Pitch("C0")
    | "sound" -> Sound((["C0"], 0., 0))
  | "intArr" -> Array([Int(0)])
  | "doubleArr" -> Array([Double(0.0)])
  | "booleanArr" -> Array([Boolean(false)])
  | "pitchArr" -> Array([Pitch("C0")])
  | "soundArr" -> Array([Sound(["C0"], 0., 0)])
  | _ -> Int(0)

(* if mixdown is not called, bytecode has x\n to indicate to BytecodeTranslator that it
shouldn't attempt to play/write MIDI *)
let stopMixDown () =
  let file = "bytecode" in
    let oc = open_out file in
      (fprintf oc "x\n";
      close_out oc)

(* global mixdown flag to see if mixdown has been called in which case we should append, not
re write a file *)
```

```
let first_mixdown_flag = ref false;;
(* default bpm value *)
let bpm = ref 120;;
let opt = ref "b";;

(* current function name *)
let fname = ref "" ;;

let run (vars, funcs) =
  (* Put function declarations in a symbol table *)
  let func_decls = List.fold_left
    (fun funcs fdecl -> NameMap.add fdecl.fname fdecl funcs)
    NameMap.empty funcs
  in
  (* Put function type in a table of function types *)
  let func_types = List.fold_left
    (fun funcs fdecl -> TypeMap.add fdecl.fname fdecl.rtype funcs)
    TypeMap.empty funcs
  in

  (* set up the function that decomposes a function call *)
  let rec call fdecl actuals globals =

    (* evals expressions and updates globals *)
    let rec eval env = function
       Int(i) -> Int(i), env
      | Double(d) -> Double(d), env
      | Boolean(b) -> Boolean(b), env
      | Pitch(p) -> Pitch(p), env
      | Sound(p,d,a) -> Sound(p,d,a), env
      (* Arrays *)
      | Array(e) ->
        let evaledExprs, env = List.fold_left
          (fun (values, env) expr ->
            let v, env = eval env expr in v::values, env)
          ([], env) (List.rev e)
        in
        (* type check *)
        (* make sure array isn't empty *)
        if evaledExprs = [] then raise (Failure(
                                "Cannot initialize empty array"))
        else
        (* traverse through the array, make sure the same*)
        let hd = List.hd evaledExprs in
        let v1Type = getType hd in
        let rec check = function
          head::tail -> let v2Type = getType head in
            if v1Type = v2Type then check tail
            else raise (Failure(v2Type^" in an array of type "^v1Type))
          | []     -> evaledExprs
        in ignore(check evaledExprs);
         Array(evaledExprs), env
      (* Index *)
      | Index(a,i) -> let v, (locals, globals) = eval env (Id(a)) in
        let rec lookup arr indices =
          let arr = match arr with
            Array(n) -> n
            | _ -> raise (Failure(a ^ " is not an array. Cannot access index"))
          in
          let index, env = eval env indices in
          (match index with
            Int(i) -> (try List.nth arr i, env with Failure("nth") -> raise
               (Failure "Index out of bounds"))
            | _ -> raise (Failure "Invalid index")
          )
        in
        lookup v (List.hd i)
      | Id(var) ->
        let locals, globals = env in
        if NameMap.mem var locals then
          (NameMap.find var locals), env
        else if NameMap.mem var globals then
          (NameMap.find var globals), env
        else raise (Failure ("undeclared identifier " ^ var))
       | Assign(var, e) ->
        (* for type check, use Index[0] as reference *)
        let lvar = (match var with
          Index(a, i) -> Index(a, [Int(0)])
          | _ -> var
        ) in

        let v1, env = eval env lvar in
        let v1Type = getType v1 in

        let v, (locals, globals) = eval env e in
        (match var with
          Id(name) ->
          (match eval env (Id(name)) with
            Index(a, i), _ -> eval env (Assign((Index(a,i), e)))
            | _,_ ->

            let v2Type = getType v in
            (* The local identifiers have already been added to ST in the first pass.
            Checks if it is indeed in there.*)
            if NameMap.mem name locals then
              begin
                (* if both are arrays, check the type of its elements instead *)
                let v1Type2 = if v1Type = "array" && v2Type = "array" then
                  (getType (match v1 with Array(v::_) -> v
                              | _ -> v1))
                else v1Type in
                let v2Type2 = if v2Type = "array" && v1Type = "array" then
                  (getType (match v with Array(d::_) -> d
                              | _ -> v))
                else v2Type in

                (* Updates the var in the ST to evaluated expression e, which is stored in v.
                Returns v as the value because this is the l-value*)
                if v1Type2 = v2Type2 then
                  v, (NameMap.add name v locals, globals)
                else raise(Failure ("type mismatch: "^v1Type2^" with "^v2Type2))
              end
            else if NameMap.mem name globals then
              begin
                (* if both are arrays, check the type of its elements instead *)
                let v1Type2 = if v1Type = "array" && v2Type = "array" then
                  (getType (match v1 with Array(v::_) -> v
                              | _ -> v1))
                else v1Type in
                let v2Type2 = if v2Type = "array" && v1Type = "array" then
                  (getType (match v with Array(d::_) -> d
                              | _ -> v))
              else v2Type in

              (* Updates the var in the ST to evaluated expression e, which is stored in v.
              Returns v as the value because this is the l-value*)
              if v1Type2 = v2Type2 then
                v, (locals, NameMap.add name v globals)
              else raise(Failure ("type mismatch: "^v1Type2^" with "^v2Type2))
            end
          else raise (Failure ("undeclared identifier " ^ name))
      )
      | Index(name, indices) ->
        let rec getIndex e =
          let v, env = eval env e in
          (match v with
            Int(i) -> i
            (*Need to call getIndexFromVar again because function needs to return only 1
             value*)
            | e ->
              print_endline (string_of_expr e);
               raise (Failure ("Illegal index"))
        )
        in
        let v2Type = (getType v) in
        let rec setElt exprs = function
          [] -> raise (Failure ("Cannot assign to empty array"))
          | hd :: [] -> let idx = getIndex hd in
            if idx < (List.length exprs) then
              let arr = (Array.of_list exprs) in arr.(idx) <- v; Array.to_list arr
            else
              let arr =
              (Array.append
                (Array.of_list exprs)
                (Array.make (1+idx-(List.length exprs))
                  (initType v2Type)))
            in
                arr.(idx) <- v; Array.to_list arr
          | _ -> raise (Failure ("Cannot assign to this array"))
        in
        if NameMap.mem name locals then
          begin
            let exprList = (match (NameMap.find name locals) with
              Array(a) -> a
              | _ -> raise (Failure (name ^ " is not an array"))) in
            let newArray = Array(setElt exprList indices) in

            if v1Type = v2Type then
              v, (NameMap.add name newArray locals, globals)
            else raise(Failure ("type mismatch: "^v1Type^" with "^v2Type))
          end
        else if (NameMap.mem name globals) then
          begin
            let exprList = (match (NameMap.find name globals) with
              Array(a) -> a
              | _ -> raise (Failure (name ^ " is not an array"))) in
            let newArray = Array(setElt exprList indices) in

            if v1Type = v2Type then
              v, (locals, NameMap.add name newArray globals)
            else raise(Failure ("type mismatch: "^v1Type^" with "^v2Type))
          end
        else
```

```
      raise (Failure (name ^ " was not properly initialized as an array"))
| _ -> raise (Failure ("Can only assign variables or array indices")))

(* binop operators *)
| Binop(e1, op, e2) ->
  let v1, env =
  (match eval env e1 with
    Index(a,i), _ ->  eval env (Index(a,i))
    | _,_ -> eval env e1) in
  let v2, env =
  (match eval env e2 with
    Index(a,i), _ -> eval env (Index(a,i))
    | _,_ -> eval env e2) in
  let v1Type = getType v1 in
  let v2Type = getType v2 in
  (match op with
    (* v1 + v2 *)
    Add -> (match v1 with
      Int(i1) -> (match v2 with
        Double(d2) -> Double (float_of_int i1 +. d2)
        | Pitch(p2) -> Pitch (intToPitch(i1 + pitchToInt p2))
        | Int(i2) -> Int (i1 + i2)
        | _ -> raise (Failure (v1Type ^ " + " ^ v2Type ^ " is not a valid operation")))
      | Double(d1) -> (match v2 with
        Int(i2) -> Double (d1 +. (float_of_int i2))
        | Double(d2) -> Double (d1 +. d2)
        | _ -> raise (Failure (v1Type ^ " + " ^ v2Type ^ " is not a valid operation")))
      | Pitch(p1) -> (match v2 with
        Int(i2) -> Pitch (intToPitch(pitchToInt p1 + i2))
        | _ -> raise (Failure (v1Type ^ " + " ^ v2Type ^ " is not a valid operation")))
      | _ -> raise (Failure (v1Type ^ " + " ^ v2Type ^ " is not a valid operation")))
    (* v1 - v2 *)
    | Sub -> (match v1 with
      Int(i1) -> (match v2 with
        Double(d2) -> Double (float_of_int i1 -. d2)
        | Pitch(p2) -> Pitch (intToPitch(i1 - pitchToInt p2))
        | Int(i2) -> Int (i1 - i2)
        | _ -> raise (Failure (v1Type ^ " - " ^ v2Type ^ " is not a valid operation")))
      | Double(d1) -> (match v2 with
        Int(i2) -> Double (d1 -. float_of_int i2)
        | Double(d2) -> Double (d1 -. d2)
        | _ -> raise (Failure (v1Type ^ " - " ^ v2Type ^ " is not a valid operation")))
      | Pitch(p1) -> (match v2 with
        Int(i2) -> Pitch (intToPitch(pitchToInt p1 - i2))
        | _ -> raise (Failure (v1Type ^ " - " ^ v2Type ^ " is not a valid operation")))
      | _ -> raise (Failure (v1Type ^ " - " ^ v2Type ^ " is not a valid operation")))
    (* v1 * v2 *)
    | Mult ->
      (* Used for the array * int and int * array operations *)
      let rec buildList ls i =
        match i with
          1 -> ls
          | _ -> ls @ (buildList ls (i-1))
      in (match v1 with
      Int(i1) -> (match v2 with
        Array(a) -> Array (buildList a i1)
        | Sound(p,d,a) -> Sound (p,float_of_int i1 *. d, a)
        | Double(d2) -> Double (float_of_int i1 *. d2)
        | Pitch(p2) -> Pitch (intToPitch(i1 * pitchToInt p2))
        | Int(i2) -> Int (i1 * i2)
        | _ -> raise (Failure (v1Type ^ " * " ^ v2Type ^ " is not a valid operation")))
    | Double(d1) -> (match v2 with
      Sound(p,d,a) -> Sound(p,d1 *. d, a)
      | Int(i2) -> Double (d1 *. float_of_int i2)
      | Double(d2) -> Double (d1 *. d2)
      | _ -> raise (Failure (v1Type ^ " * " ^ v2Type ^ " is not a valid operation")))
    | Pitch(p1) -> (match v2 with
      Int(i2) -> Pitch (intToPitch(pitchToInt p1 * i2))
      | _ -> raise (Failure (v1Type ^ " * " ^ v2Type ^ " is not a valid operation")))
    | Sound(p,d,a) -> (match v2 with
      Int(i2) -> Sound(p,d *. float_of_int i2,a)
      | Double(d2) -> Sound(p,d *. d2,a)
      | _ -> raise (Failure (v1Type ^ " * " ^ v2Type ^ " is not a valid operation")))
    | Array(a) -> (match v2 with
      Int(i2) -> Array (buildList a i2)
      | _ -> raise (Failure (v1Type ^ " * " ^ v2Type ^ " is not a valid operation")))
    | _ ->raise (Failure (v1Type ^ " * " ^ v2Type ^ " is not a valid operation")))
  (* v1 / v2 *)
  | Div -> (match v1 with
    Int(i1) -> (match v2 with
      Double(d2) -> Double (float_of_int i1 /. d2)
      | Pitch(p2) -> Pitch (intToPitch(i1 / pitchToInt p2))
      | Int(i2) -> Int (i1 / i2)
      | _ -> raise (Failure (v1Type ^ " / " ^ v2Type ^ " is not a valid operation")))
    | Double(d1) -> (match v2 with
      Int(i2) -> Double (d1 /. float_of_int i2)
      | Double(d2) -> Double (d1 /. d2)
      | _ -> raise (Failure (v1Type ^ " / " ^ v2Type ^ " is not a valid operation")))
    | Pitch(p1) -> (match v2 with
      Int(i2) -> Pitch (intToPitch(pitchToInt p1 / i2))
      | _ -> raise (Failure (v1Type ^ " / " ^ v2Type ^ " is not a valid operation")))
    | _ -> raise (Failure (v1Type ^ " / " ^ v2Type ^ " is not a valid operation")))
  (* v1 % v2 *)
  | Mod -> (match v1 with
    Int(i1) -> (match v2 with
      Int(i2) -> Int (i1 mod i2)
      | Pitch(p2) -> Pitch (intToPitch(i1 mod pitchToInt p2))
      | _ -> raise (Failure (v1Type ^ " % " ^ v2Type ^ " is not a valid operation")))
    | Pitch(p1) -> (match v2 with
      Int(i2) -> Pitch (intToPitch(pitchToInt p1 mod i2))
      | _ -> raise (Failure (v1Type ^ " % " ^ v2Type ^ " is not a valid operation")))
    | _ -> raise (Failure (v1Type ^ " % " ^ v2Type ^ " is not a valid operation")))
  (* v1 || v2 *)
  | Or ->
    if v1Type = "bool" && v2Type = "bool" then
      Boolean (getBoolean v1 || getBoolean v2)
    else raise (Failure (v1Type ^ " || " ^ v2Type ^ " is not a valid operation"))
  (* v1 && v2 *)
  | And ->
    if v1Type = "bool" && v2Type = "bool" then
      Boolean (getBoolean v1 && getBoolean v2)
    else raise (Failure (v1Type ^ " && " ^ v2Type ^ " is not a valid operation"))
  (* v1 == v2 *)
  | Eq -> (match v1 with
    Int(i1) -> (match v2 with
      Double(d2) -> Boolean (float_of_int i1 = d2)
      | Pitch(p2) -> Boolean (i1 = pitchToInt p2)
      | Int(i2) -> Boolean (i1 = i2)
      | _ -> raise (Failure (v1Type ^ " == " ^ v2Type ^ " is not a valid operation")))
    | Double(d1) -> (match v2 with
      Int(i2) -> Boolean (d1 = float_of_int i2)
      | Double(d2) -> Boolean (d1 = d2)
      | _ -> raise (Failure (v1Type ^ " == " ^ v2Type ^ " is not a valid operation")))
    | Pitch(p1) -> (match v2 with
      Pitch(p2) -> Boolean (pitchToInt p1 = pitchToInt p2)
      | Int(i2) -> Boolean (pitchToInt p1 = i2)
      | _ -> raise (Failure (v1Type ^ " == " ^ v2Type ^ " is not a valid operation")))
    | Boolean(b1) -> (match v2 with
      Boolean(b2) -> Boolean (b1 = b2)
      | _ -> raise (Failure (v1Type ^ " == " ^ v2Type ^ " is not a valid operation")))
    | _ -> raise (Failure (v1Type ^ " == " ^ v2Type ^ " is not a valid operation")))
  (* v1 != v2 *)
  | Neq -> (match v1 with
    Int(i1) -> (match v2 with
      Double(d2) -> Boolean (float_of_int i1 <> d2)
      | Pitch(p2) -> Boolean (i1 <> pitchToInt p2)
      | Int(i2) -> Boolean (i1 <> i2)
      | _ -> raise (Failure (v1Type ^ " != " ^ v2Type ^ " is not a valid operation")))
    | Double(d1) -> (match v2 with
      Int(i2) -> Boolean (d1 <> float_of_int i2)
      | Double(d2) -> Boolean (d1 <> d2)
      | _ -> raise (Failure (v1Type ^ " != " ^ v2Type ^ " is not a valid operation")))
    | Pitch(p1) -> (match v2 with
      Pitch(p2) -> Boolean (pitchToInt p1 <> pitchToInt p2)
      | Int(i2) -> Boolean (pitchToInt p1 <> i2)
      | _ -> raise (Failure (v1Type ^ " != " ^ v2Type ^ " is not a valid operation")))
    | Boolean(b1) -> (match v2 with
      Boolean(b2) -> Boolean (b1 <> b2)
      | _ -> raise (Failure (v1Type ^ " != " ^ v2Type ^ " is not a valid operation")))
    | _ -> raise (Failure (v1Type ^ " != " ^ v2Type ^ " is not a valid operation")))
  (* v1 < v2 *)
  | Lt -> (match v1 with
    Int(i1) -> (match v2 with
      Double(d2) -> Boolean (float_of_int i1 < d2)
      | Pitch(p2) -> Boolean (i1 < pitchToInt p2)
      | Int(i2) -> Boolean (i1 < i2)
      | _ -> raise (Failure (v1Type ^ " < " ^ v2Type ^ " is not a valid operation")))
    | Double(d1) -> (match v2 with
      Int(i2) -> Boolean (d1 < float_of_int i2)
      | Double(d2) -> Boolean (d1 < d2)
      | _ -> raise (Failure (v1Type ^ " < " ^ v2Type ^ " is not a valid operation")))
    | Pitch(p1) -> (match v2 with
      Pitch(p2) -> Boolean (pitchToInt p1 < pitchToInt p2)
      | Int(i2) -> Boolean (pitchToInt p1 < i2)
      | _ -> raise (Failure (v1Type ^ " < " ^ v2Type ^ " is not a valid operation")))
    | Boolean(b1) -> (match v2 with
      Boolean(b2) -> Boolean (b1 < b2)
      | _ -> raise (Failure (v1Type ^ " < " ^ v2Type ^ " is not a valid operation")))
    | _ -> raise (Failure (v1Type ^ " < " ^ v2Type ^ " is not a valid operation")))
  (* v1 > v2 *)
  | Gt -> (match v1 with
    Int(i1) -> (match v2 with
      Double(d2) -> Boolean (float_of_int i1 > d2)
      | Pitch(p2) -> Boolean (i1 > pitchToInt p2)
      | Int(i2) -> Boolean (i1 > i2)
      | _ -> raise (Failure (v1Type ^ " > " ^ v2Type ^ " is not a valid operation")))
    | Double(d1) -> (match v2 with
      Int(i2) -> Boolean (d1 > float_of_int i2)
      | Double(d2) -> Boolean (d1 > d2)
      | _ -> raise (Failure (v1Type ^ " > " ^ v2Type ^ " is not a valid operation")))
    | Pitch(p1) -> (match v2 with
      Pitch(p2) -> Boolean (pitchToInt p1 > pitchToInt p2)
      | Int(i2) -> Boolean (pitchToInt p1 > i2)
```

```
          | _ -> raise (Failure (v1Type ^ " > " ^ v2Type ^ " is not a valid operation")))
        | Boolean(b1) -> (match v2 with
          Boolean(b2) -> Boolean (b1 > b2)
          | _ -> raise (Failure (v1Type ^ " > " ^ v2Type ^ " is not a valid operation")))
        | _ -> raise (Failure (v1Type ^ " > " ^ v2Type ^ " is not a valid operation")))
      (* v1 <= v2 *)
      | Leq -> (match v1 with
        Int(i1) -> (match v2 with
          Double(d2) -> Boolean (float_of_int i1 <= d2)
          | Pitch(p2) -> Boolean (i1 <= pitchToInt p2)
          | Int(i2) -> Boolean (i1 <= i2)
          | _ -> raise (Failure (v1Type ^ " <= " ^ v2Type ^ " is not a valid operation")))
        | Double(d1) -> (match v2 with
          Int(i2) -> Boolean (d1 <= float_of_int i2)
          | Double(d2) -> Boolean (d1 <= d2)
          | _ -> raise (Failure (v1Type ^ " <= " ^ v2Type ^ " is not a valid operation")))
        | Pitch(p1) -> (match v2 with
          Pitch(p2) -> Boolean (pitchToInt p1 <= pitchToInt p2)
          | Int(i2) -> Boolean (pitchToInt p1 <= i2)
          | _ -> raise (Failure (v1Type ^ " <= " ^ v2Type ^ " is not a valid operation")))
        | Boolean(b1) -> (match v2 with
          Boolean(b2) -> Boolean (b1 <= b2)
          | _ -> raise (Failure (v1Type ^ " <= " ^ v2Type ^ " is not a valid operation")))
        | _ -> raise (Failure (v1Type ^ " <= " ^ v2Type ^ " is not a valid operation")))
      (* v1 >= v2 *)
      | Geq -> (match v1 with
        Int(i1) -> (match v2 with
          Double(d2) -> Boolean (float_of_int i1 >= d2)
          | Pitch(p2) -> Boolean (i1 >= pitchToInt p2)
          | Int(i2) -> Boolean (i1 >= i2)
          | _ -> raise (Failure (v1Type ^ " >= " ^ v2Type ^ " is not a valid operation")))
        | Double(d1) -> (match v2 with
          Int(i2) -> Boolean (d1 >= float_of_int i2)
          | Double(d2) -> Boolean (d1 >= d2)
          | _ -> raise (Failure (v1Type ^ " >= " ^ v2Type ^ " is not a valid operation")))
        | Pitch(p1) -> (match v2 with
          Pitch(p2) -> Boolean (pitchToInt p1 >= pitchToInt p2)
          | Int(i2) -> Boolean (pitchToInt p1 >= i2)
          | _ -> raise (Failure (v1Type ^ " >= " ^ v2Type ^ " is not a valid operation")))
        | Boolean(b1) -> (match v2 with
          Boolean(b2) -> Boolean (b1 >= b2)
          | _ -> raise (Failure (v1Type ^ " >= " ^ v2Type ^ " is not a valid operation")))
        | _ -> raise (Failure (v1Type ^ " >= " ^ v2Type ^ " is not a valid operation")))
      ), env

    (* !e *)
    | Not(e) ->
      let v, env = eval env e in
      let vType = getType v in
      (match v with
      Boolean(b) -> Boolean (not b), env
      | _ -> raise (Failure (vType ^ " has no ! operator")))

    (* -e *)
    | Neg(e) ->
      let v, env = eval env e in
      let vType = getType v in
      (match v with
      Int(i) -> Int (0 - i), env
      | Double(d) -> Double (0. -. d), env
      | _ -> raise (Failure (vType ^ " has no - operator")))
      | Call("setDuration", actuals) ->
        let actuals, env = List.fold_left
          (fun (actuals, env) actual ->
          let v, env = eval env actual in v :: actuals, env)
          ([], env) (List.rev actuals)
        in if (List.length actuals != 2)
        then raise(Failure("setDuration takes a sound and a double"))
        else let newDuration =
          (match (List.nth actuals 1) with
            Double(d) -> d
            | Index(a,i) ->
              (match eval env (Index(a,i)) with
                Double(d),_ -> d
                | _,_ -> raise (Failure ("Second argument must evaluate to a double")) )
            | _ -> raise (Failure ("Second argument must evaluate to a double"))
          )
        in
        (match (List.hd actuals) with
          Sound(p, d, a) -> Sound(p,newDuration,a), env
          | Index(a,i) ->
            (match eval env (Index(a,i)) with
              Sound(p,d,a),_ -> Sound(p,newDuration,a), env
              | _,_ -> raise (Failure ("First argument must be a sound")) )
          | _ -> raise (Failure ("First argument must be a sound"))
        )
      | Call("setAmplitude", actuals) ->
        let actuals, env = List.fold_left
          (fun (actuals, env) actual ->
          let v, env = eval env actual in v :: actuals, env)
          ([], env) (List.rev actuals)
        in if (List.length actuals != 2)
        then raise(Failure("setAmplitude takes a sound and an integer"))
        else let newAmplitude =
          (match (List.nth actuals 1) with
            Int(i) -> i
            | Index(a,i) ->
              (match eval env (Index(a,i)) with
                Int(i),_ -> i
                | _,_ -> raise (Failure ("Second argument must evaluate to an integer")) )
            | _ -> raise (Failure ("Second argument must evaluate to an integer"))
          )
        in
        (match (List.hd actuals) with
          Sound(p, d, a) -> Sound(p,d,newAmplitude), env
          | Index(a,i) ->
            (match eval env (Index(a,i)) with
              Sound(p,d,a),_ -> Sound(p,d,newAmplitude), env
              | _,_ -> raise (Failure ("First argument must be a sound")) )
          | _ -> raise (Failure ("First argument must be a sound"))
        )
      | Call("setPitches", actuals) ->
        let actuals, env = List.fold_left
          (fun (actuals, env) actual ->
          let v, env = eval env actual in v :: actuals, env)
          ([], env) (List.rev actuals)
        in if (List.length actuals != 2)
        then raise(Failure("setPitches takes a sound and a pitch array"))
        else let newPitches =
          (match (List.nth actuals 1) with
            Array(i) -> (* print_endline (string_of_expr (List.hd i)); *)
```

```
              List.rev (List.map (fun e -> string_of_expr e) i)
            | Index(a,i) ->
              (match eval env (Index(a,i)) with
                Array(i),_ -> List.rev (List.map (fun e -> string_of_expr e) i)
                | Pitch(p),_ -> p::[]
                | _,_ -> raise (Failure ("Second argument must be an array of pitches")) )
            | _ -> raise (Failure ("Second argument must be an array of pitches"))
          )
        in
        (match (List.hd actuals) with
          Sound(p, d, a) -> Sound(newPitches,d,a), env
          | Index(a,i) ->
            (match eval env (Index(a,i)) with
              Sound(p,d,a),_ -> Sound(newPitches,d,a), env
              | _,_ -> raise (Failure ("First argument must be a sound")) )
          | _ -> raise (Failure ("First argument must be a sound"))
        )
      | Call("print", [e]) ->
        let v, env =
          (match eval env e with
            Index(a,i),_ -> eval env (Index(a,i))
            | _,_ -> eval env e) in
        let rec print = function
          Int(i) -> string_of_int i
          | Double(d) -> string_of_float d
          | Boolean(b) -> string_of_bool b
          | Pitch(p) -> p
          | Id(i) -> let v, _ = eval env (Id(i)) in
                print v
          | Sound(p,d,a) -> "|" ^ String.concat ", " (List.rev p) ^ "|:" ^ string_of_float d ^
":" ^ string_of_int a
          | Array(a) -> let rec build = function
              hd :: [] -> (print hd)
              | hd :: tl -> ((print hd) ^ ", " ^ (build tl))
              | _ -> raise (Failure ("Item cannot be printed"))
            in
            "[" ^ build a ^ "]"
          | _ -> raise (Failure ("Item cannot be printed"))
        in
          print_endline (print v);
          Int(0), env

      | Call ("mixDown", actuals) ->
        let track_number = ref "0" in (*default track number if not specified*)
        let actuals, env = List.fold_left
          (fun (actuals, env) actual ->
            (* print_endline (Ast.string_of_expr actual); *)
          let v, env = eval env actual in v :: actuals, env)
          ([], env) (List.rev actuals)
        in
        (* Check args see if track number is specified *)
        if List.length actuals = 2 then
          begin
            (* Checks if 2nd arg is an int and if it is within its range. Then sets
track_number *)
            ignore(try (int_of_string (Ast.string_of_expr (List.hd (List.tl actuals)))) with
              Failure _ -> raise (stopMixDown(); Failure
               ("Invalid mixdown args. mixdown(<Array of Sounds or Sound>, <optional, Int,
       track_num, 0 - 15>")));
            track_number := (Ast.string_of_expr (List.hd (List.tl actuals)));
            if (((int_of_string !track_number) > 15) || ((int_of_string !track_number) < 0))
then
        raise (Failure ("Invalid track_num in mixdown. track_num should be 0 - 15"))
          end;
        if List.length actuals > 2 then
          begin
            stopMixDown();
            raise (Failure ("Invalid mixdown args. Mixdown
               (<Array of Sounds or Sound>, <optional Int trackNum>"))
          end;
        let file = "bytecode" in
        let rec writeByteCode = function
          Sound(p,d,a) -> "[" ^ String.concat ", " p ^ "]:"
               ^ string_of_float d ^ ":" ^ string_of_int a
          | Array(a) -> let rec build = function
              hd :: [] -> (writeByteCode hd)
              | hd :: tl -> ((writeByteCode hd) ^ "," ^ (build tl))
              | _  -> raise (Failure ("invalid array format"))
            in
            "[" ^ build a ^ "]"
          | _ -> raise (stopMixDown(); Failure ("argument cannot be mixdown"))
        in
          if !first_mixdown_flag = false then
            begin
              let oc = open_out file in
                fprintf oc "%s %s\n" (string_of_int !bpm) !opt;
                fprintf oc "%s\n" (!track_number ^ (writeByteCode (List.hd actuals)));
                close_out oc
            end
          else
            begin
              let oc = open_out_gen [Open_wronly; Open_creat; Open_append; Open_text] 0o666
file in
                output_string oc (!track_number ^ (writeByteCode (List.hd actuals)) ^ "\n");
                close_out oc
            end;
          (* print_endline ("Mixing down track " ^ !track_number); *)
          first_mixdown_flag := true;
          Int(0), env
      (* for pitches and sounds *)
      | Call("getAmplitude", [e]) ->
        let v, env = eval env e in
        (match v with
            Sound(p,d,a) -> Int(a), env
            | Index(a,i) ->
            (match eval env (Index(a,i)) with
              Sound(p,d,a),_ -> Int(a), env
              | _,_ -> raise (Failure ("getAmplitude can only be called on sounds")) )
          | _ -> raise (Failure ("getAmplitude can only be called on sounds"))
        )
      (* for sounds *)
      | Call("getDuration", [e]) ->
        let v, env = eval env e in
        (match v with
            Sound(p,d,a) -> Double(d), env
            | Index(a,i) ->
            (match eval env (Index(a,i)) with
              Sound(p,d,a),_ -> Double(d), env
              | _,_ -> raise (Failure ("getDuration can only be called on sounds")) )
          | _ -> raise (Failure ("getDuration can only be called on sounds"))
        )
```

```
(* for pitches and sounds *)
| Call("getPitches", [e]) ->
  let v, env = eval env e in
  (match v with
      Sound(p,d,a) ->
      let rec strings_to_pitches = function
            hd :: [] -> [Pitch(hd)]
          | hd :: tl -> [Pitch(hd)] @ strings_to_pitches tl
          | _ -> raise (Failure
                ("getPitches can only be called on sounds with pitches"))
      in
      Array(List.rev(strings_to_pitches p)), env
    | Pitch(p) -> Pitch(p), env
    | Index(a,i) ->
      (match eval env (Index(a,i)) with
        Sound(p,d,a),_ ->
        let rec strings_to_pitches = function
            hd :: [] -> [Pitch(hd)]
          | hd :: tl -> [Pitch(hd)] @ strings_to_pitches tl
          | _ -> raise (Failure
                ("getPitches can only be called on sounds with pitches"))
         in
         Array(List.rev(strings_to_pitches p)), env
        | _,_ -> raise (Failure
                ("getPitches can only be called on sounds or pitches")) )
    | _ -> raise (Failure ("getPitches can only be called on sounds or pitches"))
  )
| Call("randomInt", [bound]) ->
  let v, env = eval env bound in
  (match v with
      Int(i) -> Int(Random.int i), env
    | _ -> raise (Failure ("argument must be an int"))
  )
| Call("randomDouble", [bound]) ->
  let v, env = eval env bound in
  (match v with
      Double(d) -> Double(Random.float d), env
    | _ -> raise (Failure ("argument must be a double"))
  )
(* for arrays eyes only *)
| Call("length", [e]) ->
  let v, env = eval env e in
  (match v with
      Array(a) -> Int(List.length a), env
    | _ -> raise (Failure ("Length can only be called on arrays"))
  )
(* sets the bpm *)
| Call("bpm", actuals) ->
  let actuals, env = List.fold_left
    (fun (actuals, env) actual ->
    let v, env = eval env actual in v :: actuals, env)
    ([], env) (List.rev actuals)
  in
    if !first_mixdown_flag != false then raise (Failure
         ("Bpm must be set before mixdown is called to take affect"));
    if List.length actuals != 1 then raise (Failure
         ("One argument must be passed to bpm"));
    ignore(try (int_of_string (Ast.string_of_expr (List.hd actuals))) with
      Failure _ -> raise (Failure
         ("int must be passed to bpm. 40 - 300 is suggested")));
    bpm := (int_of_string (Ast.string_of_expr (List.hd actuals)));
```

```
      Int(0), env
  (* midi is only written and not played *)
  | Call("write", []) ->
    opt := "w";
    Int(0), env
  (* midi is only played and not written *)
  | Call("play", []) ->
    opt := "p";
    Int(0), env
  (* this does function calls. *)
  | Call(f, actuals) ->
    fname := f;
    let fdecl =
      try NameMap.find f func_decls
      with Not_found -> raise (Failure ("undefined function " ^ f))
    in
    (* get function type for initializing function return value *)
    let ftype =
      try TypeMap.find f func_types
      with Not_found -> raise (Failure ("undefined function " ^f))
    in
    let actuals, env = List.fold_left
      (fun (actuals, env) actual ->
      let v, env = eval env actual in v :: actuals, env)
      ([], env) (List.rev actuals)
    in
    let (locals, globals) = env in
    try
      let globals = call fdecl actuals globals
      in (initType ftype), (locals, globals)
    with ReturnException(v, globals) -> v, (locals, globals)
  in
  (* executes statements, calls evals on expressions *)
  let rec exec env = function
    Block(stmts) -> List.fold_left exec env stmts
    | Expr(e) -> let _, env = eval env e in env
    | If(e, s1, s2) ->
      let v, env = eval env e in
      exec env (if getBoolean v !=  false then s1 else s2)
    | While(e, s) ->
      let rec loop env =
        let v, env = eval env e in
        if getBoolean v != false then loop (exec env s) else env
      in loop env
    | For(e1, e2, e3, s) ->
    let _, env = eval env e1 in
      let rec loop env =
        let v, env = eval env e2 in
        if getBoolean v != false then
          let _, env = eval (exec env s) e3 in
          loop env
        else
          env
      in loop env
    | Loop(v, a, s) ->
      let rec runloop env idx2 = function
        [] -> env
        | hd :: tl -> let idxlist = idx2::[] in
          let locals, globals = env in
          let env =
            if NameMap.mem v locals then
            (NameMap.add v (Index(a,idxlist)) locals, globals)
          else if NameMap.mem v globals then
            (locals, NameMap.add v (Index(a,idxlist)) globals)
          else
            raise (Failure ("undeclared identifier " ^ v))
          in
        let env = exec env s in match idx2 with
          Int(i) -> runloop env (Int(i+1)) tl
          | _ -> runloop env (Int(0+1)) tl
    in
    let arr, _ = eval env (Id(a)) in
    (match arr with
      Array(x) -> runloop env (Int(0)) x
      | _ -> raise (Failure ("Looping on array was expected")))

  | Return(e) ->
  let v, (locals, globals) = eval env e in
  (* find current function type *)
  let ftype1 =
    try TypeMap.find !fname func_types
    with Not_found -> raise (Failure ("undefined function " ^(!fname)))
  in
  let ftype = (match ftype1 with
           "intArr"    -> "array"
          |"doubleArr"  -> "array"
          |"booleanArr"  -> "array"
          |"pitchArr"    -> "array"
          |"soundArr"    -> "array"
          |"boolean"    -> "bool"
          | _        -> ftype1)
  in
  let farrtype = (match ftype1 with
           "intArr"    -> "int"
          |"doubleArr"  -> "double"
          |"booleanArr"  -> "bool"
          |"pitchArr"    -> "pitch"
          |"soundArr"    -> "sound"
          |"boolean"    -> "bool"
          | _        -> ftype1)
  in

  let rtype = (getType v) in

  (* if both are arrays, check the type of its elements instead *)
  let ftype2 = if ftype = "array" && rtype = "array"
    then
      farrtype
    else ftype
  in
  let rtype2 = if ftype = "array" && rtype = "array" then
    (getType (match v with Array(d::_) -> d
                  | _ -> v))
    else rtype
  in
  (* check function type and return type *)
  if ftype2 <> rtype2 then
    raise(Failure("function type is of "^ftype2^" while the return type is of "^rtype2))
  ;
  raise (ReturnException(v, globals))
in
(* Enter the function: bind actual values to formal arguments *)
```

```
      let locals =
        try List.fold_left2
          (fun locals formal actual -> NameMap
               .add formal.paramname actual locals) NameMap.empty fdecl.formals actuals
        with Invalid_argument(_) ->
          raise (Failure ("wrong number of arguments to: " ^ fdecl.fname))
      in
      let locals = List.fold_left (* init locals to 0 *)
        (fun locals local -> NameMap
               .add local.varname (initType local.vartype) locals) locals fdecl.locals
      in
        snd (List.fold_left exec (locals, globals) fdecl.body)

    in let globals = List.fold_left
      (fun globals vdecl -> NameMap.add vdecl.varname (initType vdecl.vartype) globals)
NameMap.empty vars
    in try
      call (NameMap.find "main" func_decls) [] globals
    with Not_found -> raise (Failure("did not find the main() function"))

let _ =
  let lexbuf = Lexing.from_channel stdin in
  let program = Parser.program Scanner.token lexbuf in
  stopMixDown();
  run program
```

lullabyte

```
#!/bin/bash
#./lullabtye MyMusic.llb MyMusicMidi.mid
if [ $# -gt 0 ] && [ $# -lt 3 ]
then
        #java exits on
        touch bytecode
        echo 'x' > bytecode
        ./interpret < $1

        if [ ! -z $2 ]
        then
                java -cp jfugue-4.0.3.jar:. BytecodeTranslator bytecode $2
        else
                java -cp jfugue-4.0.3.jar:. BytecodeTranslator bytecode llb-write.mid
        fi
else
        echo "./lullabyte <*.llb> <(optional) *.mid>"
fi
```

parser.mly

```
%{ open Ast

let parse_error s = (* Called by the parser function on error *)
  print_endline s;
  flush stdout

%}

%token SEMI LPAREN RPAREN LBRACE RBRACE LBRACK RBRACK COMMA COLON PIPE
%token INT DOUBLE PITCH BOOLEAN SOUND VOID EOF
%token PLUS MINUS TIMES DIVIDE PERCENT NOT NEG
%token OR AND EQ NEQ LT GT LEQ GEQ
%token RETURN IF ELSE FOR WHILE LOOP

%token <string> ID
%token <int> INT_LIT
%token <float> DOUBLE_LIT
%token <bool> BOOLEAN_LIT
/*%token <string> SOUND_LIT*/
%token <string> PITCH_LIT
%token <string> DATATYPE
%token ASSIGN

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE
%left PERCENT
%left NOT NEG

%start program
%type <Ast.program> program

%%

program:
         /*nothing*/                {[], []}
       | program vdecl              { ($2 :: fst $1), snd $1 }
       | program fdecl              { fst $1, ($2 :: snd $1) }

fdecl:
       DATATYPE ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
       { { fname = $2;
              rtype = $1;
              formals = $4;
              locals = List.rev $7;
              body = List.rev $8 } }

formals_opt:
       /* nothing */        { [] }
       | formal_list        { List.rev $1 }

formal_list:
       param_decl { [$1] }
       | formal_list COMMA param_decl { $3 :: $1 }

param_decl:
       DATATYPE ID
              { {    paramname = $2;
                     paramtype = $1 } }

vdecl_list:
                                  { [] }
        | vdecl_list vdecl  { $2 :: $1}

vdecl:
     DATATYPE ID SEMI { {varname = $2; vartype = $1} }



/*
arr_decl:
       DATATYPE LBRACK RBRACK ID SEMI
       { { arrtype = $1;
              arrname = $4 } }
*/

stmt_list:
       /* nothing */ { [] }
       | stmt_list stmt { $2 :: $1 }

stmt:
         expr SEMI                                          {Expr($1) }
       | RETURN expr SEMI                                   { Return($2) }
       | LBRACE stmt_list RBRACE                            { Block(List.rev $2) }
       | IF LPAREN expr RPAREN stmt %prec NOELSE            { If($3, $5, Block([])) }
       | IF LPAREN expr RPAREN stmt ELSE stmt               { If($3, $5, $7) }
       | FOR LPAREN expr SEMI expr SEMI expr RPAREN stmt    { For($3, $5, $7, $9) }
       | WHILE LPAREN expr RPAREN stmt                      { While($3, $5) }
       | LOOP LPAREN ID COLON ID RPAREN stmt                { Loop($3, $5, $7) }

expr:
         INT_LIT                                            { Int($1) }
       | DOUBLE_LIT                                         { Double($1) }
       | BOOLEAN_LIT                                        { Boolean($1) }
       | PIPE pitch_list PIPE COLON DOUBLE_LIT COLON INT_LIT { Sound($2, $5, $7) }
       | PITCH_LIT                                          { Pitch($1) }
       | ID index_opt                                       { Index($1, $2) }
       | ID                                                 { Id($1) }
       | ID LPAREN actuals_opt RPAREN                       { Call($1, $3) }
       | LBRACK actuals_opt RBRACK                          { Array($2) }
       | expr ASSIGN expr                                   { Assign($1, $3) }
       | expr PLUS expr                                     { Binop($1, Add, $3) }
       | expr MINUS   expr                                  { Binop($1, Sub, $3) }
       | expr TIMES   expr                                  { Binop($1, Mult, $3) }
       | expr DIVIDE expr                                   { Binop($1, Div, $3) }
       | expr PERCENT expr                                  { Binop($1, Mod, $3) }
       | NOT expr                                           { Not($2) }
       | MINUS expr                                         { Neg($2) }
       | expr OR expr                                       { Binop($1, Or, $3) }
       | expr AND expr                                      { Binop($1, And, $3) }
       | expr EQ expr                                       { Binop($1, Eq, $3) }
       | expr NEQ expr                                      { Binop($1, Neq, $3) }
       | expr LT expr                                       { Binop($1, Lt, $3) }
       | expr GT expr                                     { Binop($1, Gt, $3) }
       | expr LEQ expr                                    { Binop($1, Leq, $3) }
       | expr GEQ expr                                    { Binop($1, Geq, $3) }
       | LPAREN expr RPAREN                               { $2 }


pitch_list:
        PITCH_LIT { [$1] }
       | pitch_list COMMA PITCH_LIT { $3 :: $1 }


actuals_opt:
       /*nothing*/    { [] }
       | actuals_list {List.rev $1}

actuals_list:
         expr                                    { [$1] }
       | actuals_list COMMA expr    { $3 :: $1 }

index_opt:
       indices { List.rev $1 }

indices:
         LBRACK expr RBRACK { [$2] }
       | indices LBRACK expr RBRACK { $3 :: $1 }
```

printAst.ml

```
open Ast

let _ =
        let lexbuf = Lexing.from_channel stdin in
        let program = Parser.program Scanner.token lexbuf in
        print_endline (Ast.string_of_program program); 1
```

runTests

```
#!/usr/bin/env ruby

argsString = ARGV.join.gsub('-', '')

if defined?(argsString) && argsString.include?("d")
	dump_flag = true
else
	dump_flag = false
end

if defined?(argsString) && argsString.include?("w")
	supress_string = " > /dev/null 2>&1"
else
	supress_string = ""
end

if defined?(argsString) && argsString.include?("o")
	show_output = true
else
	show_output = false
end

# remove junk and compile
`rm interpret > /dev/null 2>&1`
`rm printAst > /dev/null 2>&1`
`rm *.cmo > /dev/null 2>&1`
`rm *.cmi > /dev/null 2>&1`
`ocamllex scanner.mll #{supress_string}`
`ocamlyacc parser.mly #{supress_string}`
`ocamlc -i ast.ml > ast.mli`
`ocamlc -c ast.mli #{supress_string}`
`ocamlc -c ast.ml #{supress_string}`
`ocamlc -c parser.mli #{supress_string}`
`ocamlc -c scanner.ml #{supress_string}`
`ocamlc -c parser.ml #{supress_string}`
`ocamlc -c printAst.ml #{supress_string}`
`ocamlc -c helper.ml #{supress_string}`
`ocamlc -c interpreter.ml #{supress_string}`
`ocamlc -o interpret ast.cmo helper.cmo parser.cmo scanner.cmo interpreter.cmo
#{supress_string}`
`ocamlc -o printAst ast.cmo parser.cmo scanner.cmo printAst.cmo #{supress_string}`

#
# Frontend tests!
# Not totally sure on this one...
# if it gets to the AST it means everything was
# parsed correctly but I'm not sure if there could
# still be something wrong. More research required
#

puts "\n\n\e[47m\e[30mFrontend Tests:\e[0m"
# puts "\n\nFrontend Tests:"
all_pass = true
Dir['tests/*.llb'].each do |filePath|
	output = `./printAst < "#{filePath}"`

	if output == "syntax error\n"
		puts "\e[31m#{filePath}"
```

```
               all_pass = false
        end
end
if all_pass == true
        puts "\e[32mAll frontend test files were correctly parsed."
end
print "\e[0m"


#
# Backend tests!
# For each .out file in the tests directory,
# find the corresponding .llb file,
# run it,
# and compare the .out file to the actual output
#

puts "\n\n\e[47m\e[30mBackend Tests:\e[39m\e[0m"
# puts "\n\nBackend Tests:"
Dir['tests/*.out'].each do |filePath|
        expected_output = File.read(filePath)
        test_file = filePath.gsub('.out', '.llb')

        print "\e[0m"

        actual_output = `./interpret < #{test_file}`
        test_name = test_file.gsub('tests/', '')
                                              .gsub('.llb', '')

        if expected_output == actual_output
               puts "\e[32m" + test_name + " passed!\e[0m"
        elsif expected_output == actual_output && show_output
               puts "\e[32m" + test_name + " passed!\e[0m"
               puts expected_output if dump_flag == false
               puts expected_output.dump if dump_flag == true
        else
               puts "\e[31m" + test_name + " failed. \nIt could be a whitespace issue! Run with
-d to see whitespace"
               puts "\e[37m---Expected_output: \n" + expected_output if dump_flag == false
               puts "\e[31m---Actual output: \n" + actual_output if dump_flag == false
               puts "\e[37m---Expected_output: \n" + expected_output.dump if dump_flag == true
               puts "\e[31m---Actual output: \n" + actual_output.dump if dump_flag == true
        end
end

puts "\e[0m"
```

scanner.mll

```
{ open Parser }

let pitch = (['A' - 'G']('#' | 'b')?['0' - '9'] | ['C' - 'G']('#' |'b')?"10")
let int_lit = ['0'-'9']+
let dbl_lit = ['0'-'9']+['.']['0' - '9']+ | ['0'-'9']+['.'] | ['.']['0' - '9']+
let id = ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*
let int_over_int = int_lit['/']int_lit
let comma_pitch = pitch(','| ", ")
let comma_id = id(','| ", ")
let pitches = pitch | comma_pitch*pitch
let ids = id | comma_id*id
let array_of_pitches = ['|']pitches['|']
let array_of_ids = ['[']ids[']']
(*let sound = (array_of_pitches | id | array_of_ids)[':'](id | int_over_int)[':'](id |
int_lit)*)

rule token = parse
[' ' '\t' '\r' '\n']         {token lexbuf}
| "/*"                      {comment lexbuf}
| "//"                                        {comment_newline lexbuf}
| '('       { LPAREN }
| ')'       { RPAREN }
| '{'       { LBRACE }
| '}'       { RBRACE }
| ';'       { SEMI }
| ':'          { COLON }
| ','          { COMMA }
| '['       { LBRACK }
| ']'       { RBRACK }
| '='          { ASSIGN }
| '+'       { PLUS }
| '-'       { MINUS }
| '*'       { TIMES }
| '/'       { DIVIDE }
| '%'      { PERCENT }
| '!'       { NOT }
| "||"     { OR }
| "&&"     { AND }
| "=="      { EQ }
| "!="      { NEQ }
| '<'       { LT }
| '>'       { GT }
| "<="      { LEQ }
| ">="      { GEQ }
| '|'          { PIPE }

(*Types*)
| "int[]"     { DATATYPE("intArr") }
| "double[]"  { DATATYPE("doubleArr") }
| "boolean[]" { DATATYPE("booleanArr") }
| "pitch[]"   { DATATYPE("pitchArr") }
| "sound[]"   { DATATYPE("soundArr") }

| "int"     { DATATYPE("int") }
| "double"  { DATATYPE("double") }
| "boolean" { DATATYPE("bool") }
| "pitch"   { DATATYPE("pitch") }
| "sound"   { DATATYPE("sound") }
| "void"    { DATATYPE("void") }

| "true"|"false" as lxm { BOOLEAN_LIT(bool_of_string lxm)}
| "return"  { RETURN }
| "if"        { IF }
| "else"      { ELSE }
| "for"       { FOR }
| "while"   { WHILE }
| "loop"      { LOOP }

(* Type Literals must be evaluated before identifiers *)
| int_lit        as lxm { INT_LIT(int_of_string lxm) }
| dbl_lit        as lxm { DOUBLE_LIT(float_of_string lxm)}
| pitch          as lxm { PITCH_LIT(lxm)}
(*| sound              as lxm { SOUND_LIT(lxm) }*)
| id             as lxm { ID(lxm) }
| eof       { EOF }

and comment = parse
"*/"    { token lexbuf }
| _       { comment lexbuf }


and comment_newline = parse
'\n'   {token lexbuf}
| _ {comment_newline lexbuf}
```

Appendix 2.11 – Test Files

frontEndTest.llb

```
int i;
double d;
boolean b;
pitch p;

void main(int a, double b, pitch c){
	int i;
	i = 5;
	print(5);
	true = true;
	hello = world;
	false;
	A#5;
	[1, 2, 3];
	print(4.5);
}
```

test-arrays.llb

```
void main() {
	int[] a;
	int[] a2;
	int x;
	int c;
	double[] d;
	pitch[] p;
	sound[] s;
	sound s1;
	sound s2;
	sound s3;
	int[] q;
	boolean[] b;

	//Initializing and assigning to empty array. Pad the beginning with
	//type default values
//	q = [];
	q[10] = 1;
	print(1);
	print(q[10]);
	print(q);

	//Boolean arrays
//	b = [];
	b[10] = true;
	print(true);
	print(b[10]);
	print(false);
	print(b[5]);

	//Integer arrays
	a = [1, 2, 3, 4];
	print(a);
	print([5,6,7,8]);
```

```
        //Double arrays, length function, assigning beyond array length
        d = [1.0, 2.0, 3.0, 4.0];
        d[length(d)] = 5.0;
        print(length(d));
        print(d[4]);
        print(d);
        print([5.0, 6.0, 7.0, 8.0]);

        //Initializing an array with an identifier
        c = 5;
        a2 = [c, 6, 7, 8];
        print(a2);
        c = 3;
        print(a2);

        s1 = |G1|:0.25:100;
        s2 = |D2|:0.25:100;
        s3 = |F5|:0.25:100;
        s = [s1, s2, s3];
        print(s);

        //Pitch arrays
        p = [C1, D1, E1, F1];
        print(p);
        print([A5, B5, G5]);

        //Sound Arrays
        s = [|C1|:0.25:100, |D1|:0.25:100, |E1|:0.25:100, |F1|:0.25:100];
        print(s);
        print([|G5|:0.25:100, |A5|:0.25:100, |B5|:0.25:100]);

        //Accessing individual elements
        print(a[0]);
        print(d[1]);
        print(p[2]);
        print(s[3]);

        //Re-assigning individual elements
        a[0] = 10;
        x = 15;
        a[3] = x;
        d[1] = 2.5;
        print(a);
        print(d);
}
```

test-arrays.out

```
1
1
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
true
true
false
false
[1, 2, 3, 4]
[5, 6, 7, 8]
5
5.
[1., 2., 3., 4., 5.]
```

```
[5., 6., 7., 8.]
[5, 6, 7, 8]
[5, 6, 7, 8]
[|G1|:0.25:100, |D2|:0.25:100, |F5|:0.25:100]
[C1, D1, E1, F1]
[A5, B5, G5]
[|C1|:0.25:100, |D1|:0.25:100, |E1|:0.25:100, |F1|:0.25:100]
[|G5|:0.25:100, |A5|:0.25:100, |B5|:0.25:100]
1
2.
E1
|F1|:0.25:100
[10, 2, 3, 15]
[1., 2.5, 3., 4., 5.]
```

test-assign.llb

```
void main() {
       int i;
       double d;
       pitch p;
       sound s1;
       sound s2;

       i = 1;
       d = 1.0;
       p = A5;
       s1 = |A5|:0.25:100;
       s2 = s1;

       print(i);
       print(d);
       print(p);
       print(s1);
       print(s2);
}
```

test-assign.out

```
1
1.
A5
|A5|:0.25:100
|A5|:0.25:100
```

test-builtinfunctions.llb

```
void main(){
       int[] one;
       int[] two;
       int[] three;

       sound a;
       pitch b;
       sound c;

       a = |A4, A5, B6, B7|:0.5:100;
       b = B5;
       c = |C2|:5.0:1;
```

```
        one = [1];
        two = [1, 2];
        three = [1, 2, 3];

        print(getPitches(a));
        print(getPitches(b));

        print(getDuration(a));
        print(getDuration(c));

        print(getAmplitude(a));
        print(getAmplitude(c));

        print(length(one));
        print(length(two));
        print(length(three));
//      print(length([]));

        a = setDuration(a, 2.5+2.5);
        print(getDuration(a));

        a = setAmplitude(a, 45+5);
        print(getAmplitude(a));

        a = setPitches(a, [C4+1, F4+12]);
        print(getPitches(a));

        a = setPitches(a, [C4+12]);
        print(getPitches(a));

        //print(randomInt(32));


        /* print(length(1)); length can only be called on arrays*/
}
```

test-builtinfunctions.out

```
[A4, A5, B6, B7]
B5
0.5
5.
100
1
1
2
3
5.
50
[C#4, F5]
[C5]
```

test-fib.llb

```
int fib (int n){
        if (n <= 2){
                return 1;
        } else {
```

```
		return fib(n - 1) + fib(n - 2);
	}
}



sound[] offsetFib(pitch p){
	int offset;
	sound[] accumulator;
	sound s;
	int i;

	s = setAmplitude(s, 75);
	s = setDuration(s, 0.25);

	for (i = 1; i < 11; i = i + 1){
		offset = fib(i);
		s = setPitches(s, [p + offset]);
		accumulator[i-1] = s;
	}

	return accumulator;
}

void main(){
	pitch pitchC2;
	sound[] fibScale;
	sound s;

	pitchC2 = C2;
	fibScale = offsetFib(pitchC2);
	print(fibScale);
	mixdown(fibScale);
}
```

test-for.llb

```
/*
 * Tesing FOR statements.
 */

void main(){

    int i;
    int j;

    /*
     * Basic for loop
     */
    for (i = 0 ; i < 5 ; i = i + 1) {
        print(i);
    }
    print(42);

    /*
     * For within a for.
     */
    for (i = 0; i< 5; i=i+1) {
       for (j=5-i; j>0; j=j-1){
		print(i + 1);
```

```
        }
    }
}
```

test-for.out

```
0
1
2
3
4
42
1
1
1
1
1
2
2
2
2
3
3
3
4
4
5
```

test-heyjude.llb

```
void main(){

        sound FM;
        sound CM;
        sound Gm7;
        sound BbM;
        sound C7M;

        sound[] fourFs;
        sound[] fourCs;
        sound[] fourGm7s;
        sound[] fourBbMs;
        sound[] fourC7s;

        sound[] melody;

        sound[] chordProgression;
        sound[] acc;

        int decAmount;

        bpm(80);
        decAmount = 15;

        CM = |C5, E4, G3|:0.25:70;
        FM = |C5, F4, A4|:0.25:70;
        Gm7 =|G4, Bb4, D4, F4|:0.25:70;
        BbM = |Bb4, D4, F4|:0.25:70;
        C7M = |C4, E4, G3, Bb4|:0.25:70;
```

```
	chordProgression = [FM, CM, Gm7, FM, BbM, FM, CM, FM];
	acc = createQuarterNotes(chordProgression, 15);
	acc = concatArrays([|C4|:1.0:0],acc);

	melody = [|C4|:0.75:0, quart(C6),
				|A5|:0.625:100, eighth(A5),eighth(C6),eighth(D6),
				|G5|:0.75:100, eighth(G5), eighth(A5),
				quart(Bb5), |F6|:0.375:100, eighth(F6), eighth(E6),eighth(C6),
				eighth(D6), sixteenth(C6),sixteenth(Bb5),half(A5),|C4|:0.125:0,
eighth(C6),
				eighth(D6), quart(D6), eighth(D6), eighth(G6), sixteenth(F6),
eighth(E6),sixteenth(F6), eighth(D6),
				half(C6), eighth(F5), eighth(G5), eighth(A5), eighth(D6),
				quart(C6), |C5|:0.125:0, eighth(C6), eighth(Bb5), eighth(A5),
eighth(E5), eighth(F5),
				|F5|:1.0:0];

	// print(melody);
	// print(acc);
	mixDown(melody, 0);
	mixDown(acc, 1);
}

sound quart(pitch p){
	sound s;
	pitch[] pArr;
	pArr = [p];
	s = setPitches(s, pArr);
	s = setAmplitude(s, 100);
	s = setDuration(s, 0.25);
	return s;
}

sound eighth(pitch p){
	sound s;
	pitch[] pArr;
	pArr = [p];
	s = setPitches(s, [p]);
	s = setAmplitude(s, 100);
	s = setDuration(s, 0.125);
	return s;
}

sound sixteenth(pitch p){
	sound s;
	pitch[] pArr;
	pArr = [p];
	s = setPitches(s, [p]);
	s = setAmplitude(s, 100);
	s = setDuration(s, 0.0625);
	return s;
}

sound half(pitch p){
	sound s;
	pitch[] pArr;
	pArr = [p];
	s = setPitches(s, [p]);
	s = setAmplitude(s, 100);
	s = setDuration(s, 0.5);
	return s;
}

sound[] concatFour(sound s){
	return [s] * 4;
}

sound[] concatArrays(sound[] one, sound[] two){
	sound[] out;
	int i;
	int offset;
	offset = length(one);
	for (i = 0; i < length(two); i = i+1){
		one[i+offset] = two[i];
	}
	return one;
}

sound[] decrementSound(sound[] in, int amount){
	int amplitude;
	sound s;
	int count;
	int i;
	count = 0;

	amplitude = getAmplitude(in[0]);
	for (i=0; i < length(in); i = i+1){
		s = in[i];
		s = setAmplitude(s, amplitude);
		amplitude = amplitude - amount;
		in[i] = s;
	}
	return in;
}

sound[] createQuarterNotes(sound[] in, int decAmount){
	int i;
	sound[] acc;
	for (i = 0; i < length(in); i = i+1){
		acc = concatArrays(acc, decrementSound(concatFour(in[i]), decAmount));
	}
	return acc;
}
```

test-if.llb

```
/*
 * Tesing IF statements.
 */

void main(){

    int zero;
    int one;
    int two;
    boolean t;
    boolean f;

    zero = 0;
    one = 1;
```

```
two = 2;
t = true;
f = false;

/*****************************
 * Test Case:       if(true){}
 *
 * Result:          1
 */

if (true) {
    print(1);
}

/*****************************
 * Test Case:       if(false){}
 *
 * Result:          [none]
 */

if (false) {
    print(2);
}

/*****************************
 * Test Case:       if(true){} else{}
 *
 * Result:          3
 */

if (true) {
    print(3);
}
else {
    print(4);
}

/*****************************
 * Test Case:       if(false){} else{}
 *
 * Result:          6
 */

if (false) {
    print(5);
}
else {
    print(6);
}

/*****************************
 * Test Case:       if(evaluate equality){}
 *
 * Result:          true
 */

if (t == !f) {
    print(true);
}

/*****************************
```

```
     * Test Case:       if(evaluate inequality){}
     *
     * Result:          false
     */

    if (f != !f) {
        print(false);
    }

    /*****************************
     * Test Case:       evaluating a binary operator to boolean
     *
     * Result:          42
     */

    if (two > one) {
        print(42);
    }

    /*****************************
     * Test Case:       setting a value before evaluation
     *
     * Result:          69
     */

    if (f = !f) {
        print(69);
    }

    /*****************************
     * Test Case:       if(integer)
     *
     * Result:          [none]
     */

    if (two) {
        print(69);
    }

    /*****************************
     * Test Case:       if(zero)
     *
     * Result:          [none]
     */

    if (zero) {
        print(69);
    }

}
```

test-if.out

```
1
3
6
true
false
42
69
```

test-loop.llb

```
void main() {
       sound c;
       sound d;
       sound e;
       sound[] scale;
       sound s;

       double[] durs;
       double diter;
       int i;

       int[] amps;
       int aiter;

       pitch[] pits;
       pitch piter;

       c = |C1|:0.25:100;
       d = |D1|:0.25:100;
       e = |E1|:0.25:100;

       scale =[c, d, e];
       print(scale);
       loop(s : scale) {
              s = s * 3;
              print(getPitches(s));
              print(getDuration(s));
              print(getAmplitude(s));
       }
       print(scale);

       durs = [0.1, 0.2, 0.3];
       i = 0;
       loop(diter : durs) {
              scale[i] = setDuration(scale[i], diter);
              i = i + 1;
       }
       print(scale);

       amps = [25, 50, 75];
       i = 0;
       loop(aiter : amps) {
              scale[i] = setAmplitude(scale[i], aiter);
              i = i + 1;
       }
       print(scale);

       pits = [A1, B1, F1];
       i = 0;
       loop(piter : pits) {
              scale[i] = setPitches(scale[i], piter);
              i = i + 1;
       }
       print(scale);

}
```

test-loop.out

```
[|C1|:0.25:100, |D1|:0.25:100, |E1|:0.25:100]
[C1]
0.75
100
[D1]
0.75
100
[E1]
0.75
100
[|C1|:0.75:100, |D1|:0.75:100, |E1|:0.75:100]
[|C1|:0.1:100, |D1|:0.2:100, |E1|:0.3:100]
[|C1|:0.1:25, |D1|:0.2:50, |E1|:0.3:75]
[|A1|:0.1:25, |B1|:0.2:50, |F1|:0.3:75]
```

test-mixdown.llb

```
void main(){
//        sound a;
//        a = C1:0.25:100;
//        mixDown(a);

      //sound[] a2;
      //sound[] b;
      //sound[] error3;
      sound[] d;
      //sound[] c;

      //sound[] error1;
      //sound[] error2;

      //a2 = [|G4|:0.1:100, |A4|:0.1:100, |B4|:0.1:100];
      //mixDown(a2, 0);

      //b = [|G5|:0.25:100, |A5|:0.25:100, |B5|:0.25:100];
      //mixDown(b, 0);

      //properly errors
      //error1 = [|G5|:0.25:100, |A5|:0.25:100, |B5|:0.25:100];
      //mixDown(b, 16);
      //error2 = [|G5|:0.25:100, |A5|:0.25:100, |B5|:0.25:100];
      //mixDown(b, -1);
      // error3 = [|G5|:0.25:100, |A5|:0.25:100, |B5|:0.25:100];
      // mixDown(error3, 0, 3);

      //c = [|G5, B5, D5|:0.25:100, |F5, A5, C5|:0.25:100, |G5, C5, E5|:0.25:100];
      //mixDown(c, 5);

      d = [|D6|:0.083:100, |D6|:0.083:100, |D6|:0.083:100, |D6|:0.25:100, |F6, A#6|:0.25:100,
|C6, E6|:0.25:100, |D6, F6|:0.083:100, |D6|:0.083:0, |C6, E6|:0.083:100, |D6, E6|:0.75:100];
      bpm(100);
      mixDown(d);

}
```

test-nana.llb

```
//F : Eb : Bb : Bb/C : F
void main() {
    sound FM;
    sound BbM;
    sound BbCM;
    sound EbM;

    sound[] melody;
    sound[] fprog;
    sound[] ebprog;
    sound[] bbprog;
    sound[] bbcprog;
    sound[] nana;
    sound[] gfrepeat;
    sound[] nanaend;

    bpm(80);

    FM = |C5, F4, A4|:0.25:70;
    BbM = |Bb4, D4, F4|:0.25:70;
    BbCM = |Bb4, D4, F4, C4, C3|:0.25:70;
    EbM = |Eb4, G4, Bb4|:0.25:70;

    fprog = [FM];
    fprog = fprog * 4;
    ebprog = [EbM];
    ebprog = ebprog * 4;
    bbprog = [BbM];
    bbprog = bbprog * 2;
    bbcprog = [BbCM];
    bbcprog = bbcprog * 2;

    nana = [|F5|:0.5:80, |A5|:0.25:80, |C6|:0.25:80];
    gfrepeat = [|G6|:0.09375:80, |F6|:0.03125:80, |G6|:0.125:80, |F6|:0.75:80,
                |G6|:0.09375:80, |F6|:0.03125:80, |G6|:0.125:80, |F6|:0.5:80];
    nana = append(nana, gfrepeat);
    nanaend = [|D#6|:0.125:80, |D6|:0.125:80, |C6|:1.0:80];
    nana = append(nana, nanaend);
    nana = nana * 4;

    melody = append(melody, fprog);
    melody = append(melody, ebprog);
    melody = append(melody, bbprog);
    melody = append(melody, bbcprog);
    melody = append(melody, fprog);
    melody = melody * 4;

    mixDown(melody, 0);
    mixDown(nana, 1);
}

sound[] append(sound[] a, sound[] b) {
    int i;
    int j;
    int blen;

    i = length(a);
    j = 0;
    blen = length(b);

    while(j < blen) {
        a[i] = b[j];
        i = i + 1;
        j = j + 1;
    }

    return a;
}
```

test-operator.llb

```
void main()
{
	print(3+5);
	print(9-2);
	print(4*6);
	print(10/2);
	print(4%2);
	print(true || true);
	print(true || false);
	print(false && true);
	print(true && true);
	print(3<5);
	print(8<=2);
	print(19>3);
	print(3>=5);
	print(4==4);
	print(1!=2);
	print(true==true);
	print(false==true);
	print(1.2+2.5);
	print(9.0-2.4);
	print(0.5*0.5);
	print(2.5/0.25);
	print(4.4==2.4);
	print(4.000!=2.04);
	print(2.20<4.9);
	print(0.0>1.3);
	print(190.3>=190.3);
	print(100.2<=3.3);
	print(!true);
	print(-4);
	print(-4.4);
	print(2+2.2);
	print(3.5+2);
	print(3.5-2);
	print(1-.5);
	print(6.1*3);
	print(3*4.2);
	print(1/4.0);
	print(8.2/4);
	print(9.0==9);
	print(9==9.1);
	print(1.0!=1);
	print(2!=1.1);
	print(1.1<3);
	print(4<3.2);
	print(4>5.1);
	print(5.1>4);
	print(10.<=10);
	print(11<=4.1);
	print(2.3>=6);
	print(9>=8.2);
	print(1+C0);
	print(Eb5+3);
	print(60-C0);
	print(F#3-2);
	print(5*F#0);
	print(A0*10);
	print(24/C1);
	print(D6/2);
	print(100%A0);
	print(E8%10);
	print(28==E2);
	print(A#4==33);
	print(21!=A1);
	print(B5!=40);
	print(14<A#5);
	print(B4<12);
	print(2>D3);
	print(E9>40);
	print(7<=B3);
	print(A3<=2);
	print(5>=G9);
	print(G8>=6);
	print(C3==C3);
	print(C5!=C5);
	print(B1<B2);
	print(B1>B2);
	print(E5<=E6);
	print(E5>=E6);
	print(4*|C5|:.25:100);
	print(.5*|C2|:.5:100);
	print(|D4, C9|:.5:100*2);
	print(|A5, D3|:1.:100*.25);
	print([|G2, C1|:0.25:100, |D1|:0.25:100, |E1|:0.25:100, |F1|:0.25:100]*1);
	print([|G2, C1|:0.25:100, |D1|:0.25:100, |E1|:0.25:100, |F1|:0.25:100]*3);
	print([4, 6, 10]*4);
	print([3., .5]*2);
	print(1*[|G2, C1|:0.25:100, |D1|:0.25:100, |E1|:0.25:100, |F1|:0.25:100]);
	print(3*[|G2, C1|:0.25:100, |D1|:0.25:100, |E1|:0.25:100, |F1|:0.25:100]);
	print(4*[4, 6, 10]);
	print(2*[3., .5]);
}
```

test-operators.out

```
8
7
24
5
0
true
true
false
true
true
false
true
false
true
```

```
true
true
false
3.7
6.6
0.25
10.
false
true
true
false
true
false
false
-4
-4.4
4.2
5.5
1.5
0.5
18.3
12.6
0.25
2.05
true
false
false
true
true
false
false
true
true
false
false
true
C#0
F#5
C5
E3
F#2
F#7
D0
C#3
C#0
C0
true
false
false
true
true
false
false
true
true
false
false
true
true
false
true
```

```
false
true
false
|C5|:1.:100
|C2|:0.25:100
|D4, C9|:1.:100
|A5, D3|:0.25:100
[|G2, C1|:0.25:100, |D1|:0.25:100, |E1|:0.25:100, |F1|:0.25:100]
[|G2, C1|:0.25:100, |D1|:0.25:100, |E1|:0.25:100, |F1|:0.25:100, |G2, C1|:0.25:100,
|D1|:0.25:100, |E1|:0.25:100, |F1|:0.25:100, |G2, C1|:0.25:100, |D1|:0.25:100, |E1|:0.25:100,
|F1|:0.25:100]
[4, 6, 10, 4, 6, 10, 4, 6, 10, 4, 6, 10]
[3., 0.5, 3., 0.5]
[|G2, C1|:0.25:100, |D1|:0.25:100, |E1|:0.25:100, |F1|:0.25:100]
[|G2, C1|:0.25:100, |D1|:0.25:100, |E1|:0.25:100, |F1|:0.25:100, |G2, C1|:0.25:100,
|D1|:0.25:100, |E1|:0.25:100, |F1|:0.25:100, |G2, C1|:0.25:100, |D1|:0.25:100, |E1|:0.25:100,
|F1|:0.25:100]
[4, 6, 10, 4, 6, 10, 4, 6, 10, 4, 6, 10]
[3., 0.5, 3., 0.5]
```

test-practice.llb

```
void main(){
       sound c;
       sound d;
       sound e;
       sound[] scale;
       sound s;
       double[] durs;
       double diter;
       int i;


       c = |C1|:0.25:100;
       d = |D1|:0.25:100;
       e = |E1|:0.25:100;

       durs = [0.1, 0.2, 0.3];
       i = 0;
       scale =[c, d, e];
       print(scale);
       /*loop(diter : durs) {
              scale[i] = setDuration(scale[i], diter);
              i = i+1;
       }*/

       loop(s : scale) {
              s = setDuration(s, 0.5);
              print(getPitches(s));
              print(getDuration(s));
              print(getAmplitude(s));
       }
       print(scale);
}
```

test-randomScales.llb

```
sound[] concatArrays(sound[] one, sound[] two){
        sound[] out;
        int i;
        int offset;
        offset = length(one);
        for (i = 0; i < length(two); i = i+1){
                one[i+offset] = two[i];
        }
        return one;
}

sound[] randomBluesWalk(pitch p, int n, double d){
        sound[] acc;
        int[] blues;
        sound s;
        int i;
        int r;
        pitch newPitch;

        s = setPitches(s, [p]);
        s = setDuration(s, d);
        s = setAmplitude(s, 75);

        blues = [0, 3, 5, 6, 7, 10];

        for (i = 0; i < n; i = i + 1){
                r = randomInt(4);
                newPitch = p + blues[r];
                s = setPitches(s, [newPitch]);
                acc[i] = s;
        }
        return acc;
}

sound[] staccato(sound[] s){
        double initDur;
        double finalDur;
        double restTime;
        sound rest;
        sound note;
        int i;
        int outCount;
        sound[] out;

        for (i = 0; i < length(s); i = i+1){
                initDur = getDuration(s[i]);
                finalDur = initDur*0.1;
                restTime = initDur - finalDur;

                s[i] = setDuration(s[i], finalDur);
                rest = setDuration(rest, restTime);
                out[outCount] = s[i];
                out[outCount+1] = rest;
                outCount = outCount +2;
        }



        // print(s);
        // print(rest);
        print(out);
        return out;
```

```
}



void main (){
	sound[] a;
	sound C7M;
	sound FM;
	sound[] FthenC;
	sound[] high;
	sound[] low;
	sound[] acc;

	bpm(100);

	high = [|C6|:0.125:100, |G6|:0.125:100, |Bb6|:0.125:100];
	low = [|C3|:0.125:100, |G3|:0.125:100, |Bb2|:0.125:100];
	high = staccato(high);
	acc = concatArrays(acc, low);
	acc = concatArrays(acc, low);
	acc = concatArrays(acc, high);
	acc = concatArrays(acc, high);
	acc = concatArrays(acc, low);
	acc = concatArrays(acc, low);
	acc = concatArrays(acc, high);
	acc = concatArrays(acc, high);

	FM = |F5, A5, C5|:1.0:50;
	C7M = |C3, G4|:0.25:50;
	FthenC = concatArrays([FM]*2, [C7M]*8);
	// print(FthenC);
	// mixdown(acc);
	mixDown(FthenC*2, 0);



	a = randomBluesWalk(C3, 8, 0.25);
	a = concatArrays(a, randomBluesWalk(C5, 16, 0.125));
	a = concatArrays(a, randomBluesWalk(C3, 8, 0.25));
	a = concatArrays(a, randomBluesWalk(C5, 8, 0.25));
	a = concatArrays(a, staccato(randomBluesWalk(C5, 8, 0.25)));




	// print(a);
	mixDown(a, 1);
}
```

test-return.llb

```
void main() {
	print(retIntLit());
	print(retIntVar());

	print(retDblLit());
	print(retDblVar());

	print(retPitchLit());
	print(retPitchVar());
	print(retSoundLit());
	print(retSoundVar());
}

int retIntLit() {
	return 1;
}

int retIntVar() {
	int i;
	i = 1;
	return i;
}

double retDblLit() {
	return 1.5;
}

double retDblVar() {
	double d;
	d=1.5;
	return d;
}

pitch retPitchLit() {
	return A5;
}

pitch retPitchVar() {
	pitch p;
	p = A5;
	return p;
}

sound retSoundLit() {
	return |A5|:0.25:100;
}

sound retSoundVar() {
	sound s;
	s = |A5|:0.25:100;
	return s;
}
```

test-return.out

```
1
1
1.5
1.5
A5
A5
|A5|:0.25:100
|A5|:0.25:100
```

test-vars.llb

```
int i_glob;
```

```
double d_glob;
boolean b_glob;
pitch p_glob;
sound s_glob;

void main() {
	i_glob = 1;
	d_glob = 1.0;
	b_glob = true;
	p_glob = A1;
	s_glob = |A1|:0.25:100;

	//Globals
	print(i_glob);
	print(d_glob);
	print(b_glob);
	print(p_glob);
	print(s_glob);

	//Locals
	printlocs();
}

void printlocs() {
	int i_loc;
	double d_loc;
	boolean b_loc;
	pitch p_loc;
	sound s_loc;

	i_loc = 2;
	d_loc = 2.0;
	b_loc = false;
	p_loc = A2;
	s_loc = |A2|:0.125:50;

	print(i_loc);
	print(d_loc);
	print(b_loc);
	print(p_loc);
	print(s_loc);
}
```

test-vars.out

```
1
1.
true
A1
|A1|:0.25:100
2
2.
false
A2
|A2|:0.125:50
```

test-while.llb

```
/*
```

```
 * Tesing WHILE statements.
 */

void main() {
    int i;
    int n;
    boolean b;

    /*
     * Increment a counter and exit at 5.
     */
    i = 0;
    while(i < 5) {
        print(i);
        i = i + 1;
    }
    print(42);

    /*
     * Loop until the boolean is set to false.
     */
    b = true;
    while(b) {
        if(i == 10) {
            b = false;
        }
        print(i);
        i = i + 1;
    }

    /*
     * While within a while.
     */
    i = 0;
    while (i <= 5){
        n = i;
        while (n >=0){
            print(n);
            n = n - 1;
        }
        i = i + 1;
    }
}
```

test-while.out

```
0
1
2
3
4
42
5
6
7
8
9
10
0
1
```

```
0
2
1
0
3
2
1
0
4
3
2
1
0
5
4
3
2
1
0
```

Appendix 3 - Log of Git Commits

```
commit 82b3cafea3736a5d68143cc2d01585d8b3940540
Author: 2-win7-40g <peter100g@ubuntu.(none)>
Date:   Fri Dec 20 11:08:34 2013 -0500

    added nana part of hey jude

commit 5a23f5125f5c857eebd239c2ad217f3fafeb4db8
Merge: eeb9165 d591274
Author: tehapple <stanley0856@gmail.com>
Date:   Fri Dec 20 00:40:20 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit eeb91659f9090d750580bcefe9a288e3c91f2566
Author: tehapple <stanley0856@gmail.com>
Date:   Fri Dec 20 00:40:07 2013 -0500

    fixed bug and ambiguity of boolean and bool arrays; hot
fix

commit d591274db1bc719e91722afd93e3edc6ba502727
Merge: 0de1f53 e897db2
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Fri Dec 20 00:20:19 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 0de1f5355e59b47374f3f60c870981c1b2a89e69
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Fri Dec 20 00:18:40 2013 -0500

    Blew away test-moreInterestingDemo.llb

commit e897db24f7e154ded669e829ba0e9fe85d2ab9d7
Merge: 8166dfa 6b0e6bd
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Fri Dec 20 00:09:32 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 8166dfa793bf80bec634472573fa0668a3268202
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Fri Dec 20 00:09:17 2013 -0500

    moved function definitions below main

commit 6b0e6bd668203ed894d12fd21ccd14f8f6b6a204
Merge: 337fbc9 0b0faca
Author: tehapple <stanley0856@gmail.com>
Date:   Fri Dec 20 00:08:42 2013 -0500

    fixed merge conflict

commit 337fbc9344b4f796c13ab00c054d2b6c0000f322
Author: tehapple <stanley0856@gmail.com>
Date:   Fri Dec 20 00:06:26 2013 -0500

    saves functions types in a module; creates global
function name; type check function type with return type

commit 0b0facaae1e981a09ad2a3fe6b602b1ec1957ee3
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Fri Dec 20 00:00:38 2013 -0500

    Blew away outdated interpret.ml and improved Makefile

commit da653607298eea619bc103cd9f1c5a377f7d9532
Merge: 850a904 390dfde
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 23:57:02 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 850a904ed36a309a9aaab11cebb520d5f8c37024
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 23:56:52 2013 -0500

    mixDown not mixdown

commit 390dfdece003033861a90de82e1e6a99aeff88d4
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Thu Dec 19 23:50:56 2013 -0500

    Changed make clean to remove unnecessary files

commit 820d9f5722846161818953547d9b9e0d9d1bece4
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Dec 19 23:44:46 2013 -0500

    changed default bpm and removed dead code

commit 39c297f24a28c293221cbc7d84d206ba6723a4c8
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 22:47:17 2013 -0500

    fixed error catch from parse error for BytecodeTranslator

commit 9d23768c3e0df4ff40e6783ff8613cfb378b897f
Merge: 9c0e827 05f754e
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 22:03:51 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 9c0e827b39af00445c611b99cae55c092820009c
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 22:02:47 2013 -0500

    all green tests now

commit 05f754e21796cd5b482d1aecb33cddc889a4fd8e
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 22:00:26 2013 -0500
```

```
    checks for #arguments 1/2 only

commit b3dc222d9ed388c834955ae814b56d87df91cf0c
Merge: 7c9473f 4beb469
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Dec 19 21:43:28 2013 -0500

    Merge branch 'master' of
github.com:peterxu422/PLTCompiler

commit 7c9473fb7ad6b93d0c2c95d880fee797e246f0a3
Merge: aefeaaa 894641f
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Dec 19 21:41:58 2013 -0500

    Merge branch 'master' of
github.com:peterxu422/PLTCompiler

commit aefeaaa66aa8fdc8b68a930cfe6870e7d07c1fdb
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Dec 19 21:41:55 2013 -0500

    renamed mixdown to mixDown

commit 4beb469009c1dcc398f74a5b61177ba5d1907ed6
Merge: 79b41ae 894641f
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 21:41:53 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 894641f784f5aa2e63255a4a7bccdb758712714f
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 21:41:00 2013 -0500

    weird \n missing in tests/test-loop.out; fixed

commit 79b41ae5a579971cd8b7bcd6fe3ab842306697d5
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 21:39:35 2013 -0500

    wrapped this whole thing up with a pretty bow, removed
some old files

commit 95246aad1486fb0bc11fa47ec77fba9ee4899cc5
Merge: f603b2b 13c5187
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 21:37:28 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit f603b2b3f8c8bf9f744fd98414a0e83b24760aaf
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 21:37:06 2013 -0500

    fixed bug in scanner.mll (bool and boolean mixup) fixed
getType bug to eval function twice

commit 13c5187915434a66c4004feb8795ec7f6b5ae1ff
Merge: 26d149c 22c3621
Author: Peter Xu <peterxu422@gmail.com>
Date:   Thu Dec 19 21:27:06 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 26d149cb8c0b1fd7de0719e0e58c5edeccab3671
Author: Peter Xu <peterxu422@gmail.com>
Date:   Thu Dec 19 21:26:55 2013 -0500

    fixed bug in loop that andrew found

commit 22c362102c3461c6e88e76335bbc533e77ed5653
Merge: 3a617ef 98c55b0
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 19:47:26 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 0bd85b4b2dea9928ddbc0a364dea347912fcab73
Merge: 9189135 98c55b0
Author: Peter Xu <peterxu422@gmail.com>
Date:   Thu Dec 19 19:47:07 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 3a617efcac79425642d6a6ce06907ce5c0d77008
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 19:47:03 2013 -0500

    added play and write

commit 9189135d22067b2ffa3a54d6b93d53724e33c220
Author: Peter Xu <peterxu422@gmail.com>
Date:   Thu Dec 19 19:43:46 2013 -0500

    modified test-arrayout

commit 98c55b018657ed40702f09d962d65cc8b9055b05
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 18:42:59 2013 -0500

    fixed warning 10s using ignore; experimenting with
tests/test-if.llb

commit b67c2fd93ec3dc48616cbd8539c607325c3aaa41
Merge: 929e9a1 e91f061
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 18:33:06 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit e91f061f4d79498fb1e6dfa96871e318f924b570
Merge: fc00c93 284a559
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 17:19:09 2013 -0500
```

```
    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit fc00c93e915dd4abe9321c7789d03ad243bd771d
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 17:18:48 2013 -0500

    pass midi file name argument

commit 8a3f1581033334a280c9d33083b8a6207d1ec8fd
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 17:18:07 2013 -0500

    pass file name argument for midi file

commit 929e9a1fd2a10ae27c533bca85e36b453fafe4e9
Merge: 9454ecb 284a559
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 17:03:27 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 284a559375e10d937c935c40452934088dfb0505
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Thu Dec 19 16:55:55 2013 -0500

    No more warning 8: pattern matching not exhaustive

commit 9454ecb2aae2b39b8e9b5e42210aac39ca68ba56
Merge: cd839af 4d26077
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 16:49:02 2013 -0500

    fixed merge error

commit cd839af4a1747db7b5e61996bfeea2e323f21137
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 16:46:42 2013 -0500

    fixed a warning at line 714

commit 4d26077e7eb9ce8c22df00676be4a11817616534
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Thu Dec 19 16:45:38 2013 -0500

    removed more warning 8s

commit 796dad785e43b3c2275c4fe5ee228d2c6505d8c5
Merge: bbf7941 4a82cbf
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Thu Dec 19 16:39:26 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit bbf79417ff1d567b44e9befa3252000f07e79e99
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Thu Dec 19 16:39:21 2013 -0500

    fixed more warning 8s (pattern matching non-exhaustive

commit 4a82cbf064058dfef720e604f3ddb81f0c0d1775
Merge: e6fb46a 518a648
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 16:37:25 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit e6fb46aeb2508a6d42c9884b094562dfea9bfe69
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 16:37:19 2013 -0500

    when mixdown fails java will not try to play mixdown or
write a file

commit 518a6483c94f33bb6f8058098ebfcb7bb17e13ac
Merge: 086a2de 70ce569
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Dec 19 16:36:15 2013 -0500

    Merge branch 'master' of
github.com:peterxu422/PLTCompiler

commit 086a2de867331c55ed75b2b60a779278952c239b
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Dec 19 16:36:06 2013 -0500

    fixed tests again

commit 70ce569562bd3b1d4602735f944564de6051860c
Merge: 7710a8b 319b41a
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 16:35:01 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 7710a8bb6ada5323888e1dbeb3192d6f243716c7
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 16:34:40 2013 -0500

    added raise failure for initializing empty array

commit 319b41a5d5c797a4b96dd38eff699819b9171569
Merge: a25e663 88f5227
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Dec 19 16:32:23 2013 -0500

    Merge branch 'master' of
github.com:peterxu422/PLTCompiler

commit a25e6633e8b2e17ea40377145f8d463a573616af
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Dec 19 16:32:07 2013 -0500

    fixed test suite

commit 88f522753903519d5e72f9d1c3a055e3a5242a7a
Merge: f14f4ac 49241b4
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 16:28:40 2013 -0500
```

```
    fixed merge errors

commit f14f4ac50794dba259163861004e812d0ca53abb
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 16:25:52 2013 -0500

    fixed bug in test-loop.llb; fixed int[] = 3 typecheck

commit 49241b4b773e2068835acebe1c1f462ce87c4826
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Thu Dec 19 16:24:42 2013 -0500

    removed another warning 8

commit 13bf765d78825bf185f313b85277007177a01159
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Thu Dec 19 16:02:08 2013 -0500

    eliminated 2 patternmatching not exhaustive warnings

commit cc308c20529c6d8cb37edbd9c00ca78fffb2d499
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 02:18:50 2013 -0500

    bpm works and is tested

commit 135374a1670886bc16c2b6480f1ce4a6eeaa6119
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 01:42:38 2013 -0500

    mini commit for bpm

commit 0288cf3ce7bdac31537e7b2a05e10e783f291d46
Merge: 7347db1 24e2f70
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 01:40:08 2013 -0500

    fix merge

commit 7347db1d7a685d3c7ad1fce1d8c6edfa1c7b7536
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 19 01:38:38 2013 -0500

    bpm is working

commit 24e2f700902b26d9fd42acf42178afb7a30981d9
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 01:02:59 2013 -0500

    added comments for typecheck, typecheck is working
properly

commit 5d14d035356a2a09de8ceb9fcfe1b89edf39eca5
Merge: 2e7ccec 1a1725d
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 00:47:09 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 2e7ccec71b629e6ea4dd88ee541d99caa2240172
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 00:46:37 2013 -0500

    changed llb test files because int[] x = [] is no longer
allowed

commit c71fb9d7e58c6f8123df7585ad4c89fe772fb53b
Author: tehapple <stanley0856@gmail.com>
Date:   Thu Dec 19 00:38:15 2013 -0500

    type check seems to be complete, needs to be tested

commit 1a1725df6d3c867a7a490e5b2c59b3be2dc209ee
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Dec 19 00:16:17 2013 -0500

    changed getPitch to getPitches, it's more accurate

commit b4bb19b7a381bcb456785074a8750e0c60101b5b
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Wed Dec 18 23:29:04 2013 -0500

    added printAst and update gitignore

commit 7dda6f8ae9649cb4ed60abafabbc579a95bf29a6
Merge: 9d8d96e c5221d6
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Wed Dec 18 23:24:54 2013 -0500

    Merge branch 'master' of
github.com:peterxu422/PLTCompiler

commit 9d8d96e3d0e60b30715fb83d1a3d146d372540e9
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Wed Dec 18 23:24:21 2013 -0500

    Added test programs

commit c07ba36db40e72b801738d266fa6bf305712f5fb
Merge: 2a0f7b9 9c9de23
Author: tehapple <stanley0856@gmail.com>
Date:   Wed Dec 18 22:11:17 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 2a0f7b98d0be7d988cc5ec0fe1afd4dd6f2c801e
Author: tehapple <stanley0856@gmail.com>
Date:   Wed Dec 18 22:02:49 2013 -0500

    premature push?

commit 9c9de2365b18cc96d78c6b549e54d8b2f321cdd1
Merge: c5221d6 9eaa9a5
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Wed Dec 18 21:20:18 2013 -0500

    merged conflicts in make2.sh

commit c5221d636cddd3000822e55c544358e992f19f1d
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Wed Dec 18 21:18:34 2013 -0500
```

```
    added -w -8 args to supress Warning 8: Pattern Matching is not Exhaustive

commit 9eaa9a572d9aa6a5f04408cecc089c89641f8a52
Merge: e4b51c7 f36f267
Author: tehapple <stanley0856@gmail.com>
Date:   Wed Dec 18 20:47:30 2013 -0500

    fixed merge conflict

commit e4b51c7b1842b014d9d0a8961e5c13318f0e59e7
Author: tehapple <stanley0856@gmail.com>
Date:   Wed Dec 18 20:45:47 2013 -0500

    type checking except for int[] a = [1, 2, true] case

commit f36f2674e1806e5f94bb490c240dfa95be7257b8
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Wed Dec 18 20:28:50 2013 -0500

    removed tie operator

commit 412cb17a0c236c208f996a3f4c4e4560b4d95554
Merge: 11fc151 98ef00a
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Wed Dec 18 20:21:06 2013 -0500

    merged conflict

commit 11fc15115e4242c5c4199ad1138a8e474de06262
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Wed Dec 18 20:10:26 2013 -0500

    added commenting to interpreter.ml

commit 26353a6271e12fc9e94cb0dbbb119d0802162f54
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Wed Dec 18 19:55:51 2013 -0500

    eliminated need for getInt() and removed the function from interpreter.ml

commit e0288b2716ccf4fc8f72b27d9de60e201662c275
Merge: 9cba7f0 98ef00a
Author: tehapple <stanley0856@gmail.com>
Date:   Wed Dec 18 19:55:42 2013 -0500

    fixed merge conflict

commit 9cba7f0c795ff55c893428f3029e3a0e288385ad
Author: tehapple <stanley0856@gmail.com>
Date:   Wed Dec 18 19:53:54 2013 -0500

    preminary type check; got around out of bounce error

commit 98ef00a9217ad0bd6d6a497563c1ce0c94f550fc
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Wed Dec 18 19:42:25 2013 -0500

    array indices are eval'd now

commit d5424a0de60f0b44dce7354295f55be4cc14637d
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Wed Dec 18 19:22:21 2013 -0500

    removed nathan's name from all the tests

commit 169c108a3c755555282cf09fbbae2bac3084968b
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Wed Dec 18 19:17:43 2013 -0500

    caught exception in bytecode translator and got rid of random print_endline

commit 2fbfcca39d775758c84f9dfe33364d41563bf57a
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Wed Dec 18 18:57:00 2013 -0500

    fixed test

commit 682a65e12ae9ec3b1b375fe413a64124debbc615
Merge: 6f01bd5 18582c7
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Wed Dec 18 18:54:41 2013 -0500

    Merge branch 'master' of github.com:peterxu422/PLTCompiler

commit 6f01bd5dd983029a2311ccdb5edff315dfbbee01
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Wed Dec 18 18:54:36 2013 -0500

    added randomInt and randomDouble functions

commit 18582c716167442cf0995debeb5d9656dee50213
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Wed Dec 18 18:50:07 2013 -0500

    moved buildList function to more appropriate location and changed Array() implementation back to how Andrew had it

commit 09fd9d6cfb4f1540721cfd51442fb846e5e92923
Merge: b6154c5 655c212
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Wed Dec 18 18:36:02 2013 -0500

    Merge branch 'master' of https://github.com/peterxu422/PLTCompiler
    Andrew pushed some code.  we're going to merge them.

commit b6154c5cf8bf39dd171fdbe07fe96867b9c58f2f
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Wed Dec 18 18:35:55 2013 -0500

    int * array operator works and tests added that pass

commit 3746a1717942f13bb39a8e164e3fdf67de05a9ad
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Wed Dec 18 18:31:54 2013 -0500

    added functionality to int * array operation

commit 1239f01b8af84e56bce095265381e95806d84491
```

```
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Wed Dec 18 18:25:41 2013 -0500

    added tests for array * int operation.  tests pass

commit 78c0742948cfe565c3a67f0f19f25cf3fc3e7b3a
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Wed Dec 18 18:17:34 2013 -0500

    added specified functionality for array * int operation
and fixed print to print pitches in order in a sound

commit 655c2124868b414a7b25d103121090550a258f9a
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Wed Dec 18 18:02:15 2013 -0500

    fixed tests, test suite is green

commit 1f3bbb56e7d48ba678f096af9b96994b524498ae
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Wed Dec 18 17:55:55 2013 -0500

    Added setpitch and setAmplitude and tests

commit f58c115ea41b56e0e8b7218551c8c70021f3338c
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Wed Dec 18 17:03:35 2013 -0500

    added setDuration

commit 5ab7c70fa62cfd448cd25bf38e68af8119573ffa
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Wed Dec 18 15:59:15 2013 -0500

    removed unnecessary getDouble(), getPitch(), and
getSound() functions from file

commit f5561a46cc3b6d409c5260f82986724501f786e8
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sun Dec 15 19:04:41 2013 -0500

    updated array, assign, and vars out files

commit f38e3332a05c885421238e2b35cbfaa3ed8efc4c
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sun Dec 15 19:04:05 2013 -0500

    added loop test

commit d38de661a30016953d118f3fa3c9fb197786c7c9
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sun Dec 15 18:54:41 2013 -0500

    Loop variable updates the array when assigned to and
gives the value when referenced

commit 12bc7fcc6c304d394128c58091680a926a0f25d3
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sat Dec 14 05:00:09 2013 -0500

    implemented basic loop statement

commit 08d34c0d8303018f4158fd5ffa196180e5cc7314
Merge: f51ec36 6e7d363
Author: Peter Xu <peterxu422@gmail.com>
Date:   Fri Dec 13 21:56:48 2013 -0500

    merged conflicts for array init with id

commit f51ec3693e36e013e4e7bdbb5f95434e1f64663d
Author: Peter Xu <peterxu422@gmail.com>
Date:   Fri Dec 13 21:11:32 2013 -0500

    adjusted array and vars test files

commit 2afb2fd662931c3c20465b8f251fc48b8994a811
Author: Peter Xu <peterxu422@gmail.com>
Date:   Fri Dec 13 21:10:29 2013 -0500

    started on loop and array initialization with identifiers

commit 6dac9e14da087505ad83397e8c8bbc34d0a910ee
Author: Peter Xu <peterxu422@gmail.com>
Date:   Fri Dec 13 21:09:42 2013 -0500

    reverted makefile

commit 6e7d363907656c9454cbcabdd5ab70acf1254dee
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Fri Dec 13 19:56:19 2013 -0500

    expressions are evaled when doing array init

commit 9bc13305b7e53aebf343111aad4bfdfdf700bb6c
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Fri Dec 13 16:58:42 2013 -0500

    Beefed up test-for

commit bd28f69518edaf2f238d158c6b9f39ac0fce027e
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Fri Dec 13 16:50:36 2013 -0500

    Beefed up test-while

commit 90cfbf72bedac8d7d5ed9b05b14145b2a957f233
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Fri Dec 13 15:55:30 2013 -0500

    Removed outdated test-if file

commit 3fd3ed824580e1f74f626ab7a2d3d29f39ea8075
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Fri Dec 13 15:50:27 2013 -0500

    Beefed up test-if

commit faac9fc5430488062093be2344392eac84ce4437
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Fri Dec 13 15:10:12 2013 -0500

    Fixed test-return.out

commit 1a61679d30c014bd075361cad9c92abfbc3b47a3
```

```
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Fri Dec 13 14:57:59 2013 -0500

    Test commit

commit 8685ea5cc67739e70ef63a419bd0adef5c790e2f
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Fri Dec 13 00:14:26 2013 -0500

    Mixdown can now take a in the specified track_num eg.
mixdown(a, 2) appends to track two. mixdown can be
called many times

commit 30a52caeef37f4b334825e89d91ff64f5d8f7eae
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Mon Dec 9 21:27:05 2013 -0500

    all relational operator and unary operator code refactored
in interpreter.ml

commit aa9ec0e8e66a9abe1f8a833922ed763ca73078a3
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Mon Dec 9 21:11:34 2013 -0500

    refactored == and != operator code in interpreter.ml

commit 88ad1325d103f18e145b2b8f7818d6b63f496628
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Mon Dec 9 20:45:30 2013 -0500

    Refactored Mod operator in interpreter.ml

commit 27e90b0014e4f18d9595dca0c4f8a40d610f4282
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Mon Dec 9 20:38:52 2013 -0500

    Refactored Div operator code in interpreter.ml

commit 409611e675cb79b6af4505dc22b8e261dda83708
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Mon Dec 9 20:31:24 2013 -0500

    Refactored Sub operator

commit 7e69be14c224a9401edc3b4231c5d8714c308c15
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Mon Dec 9 20:12:54 2013 -0500

    refactored Mult and Add operators

commit 629399a792ba1040f760b6202ccd49eb157cc26b
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Sun Dec 8 17:38:25 2013 -0500

    deleted unnecessary comments

commit c197fea8fe57ca4b02a85bea56bc0b3401370f6b
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Sun Dec 8 17:33:01 2013 -0500

    Added int * sound, sound * int, double * sound, sound *
double operators and tests

commit 63de71f5548a2605bb1e107453a62b66a1002a21
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Fri Dec 6 18:41:57 2013 -0500

    Any operations that can be done on ints can be done on
an int and a pitch.  pitch on pitch relational operators also
work.  tests added to test-operators.llb and test-
operators.out. added lines to compile Ben's helper.ml and
link interpret to helper.cmo in runTests and make2.sh

commit f41431ea7fc4432a2ee67d479bf5e36565c095c5
Merge: c742f16 de9dd35
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Fri Dec 6 14:51:30 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit de9dd357cb9aab79e9a924a3c5dc1f9918a970c4
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Thu Dec 5 21:38:53 2013 -0500

    fixed test-operators.out

commit c742f16fc7ac501dc713a83bdfd09b918a102735
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Thu Dec 5 21:25:37 2013 -0500

    adding helper with pitch mappings maybe other helpers
in interpret can go in here

commit 539794539af700d02019bbb7712961ef03bea39e
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Thu Dec 5 21:11:18 2013 -0500

    scanner now accepts 4., .5 as Double, added functionality
for all int op double and double op int operations

commit fa644713980a8846b4647c383173518b1f2e5abe
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Thu Dec 5 18:37:12 2013 -0500

    cleaned up operators tests

commit 0e314782c7108596fca00ec5e9eed331d6dbb528
Merge: 41c3bee c8db72f
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Thu Dec 5 18:32:15 2013 -0500

    merged conflict

commit 41c3beefc025bfe4f962eb2192a1285ef1c8621e
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Thu Dec 5 18:27:01 2013 -0500

    for some reason these didn't go through

commit c8db72feef4aea5e177c5a51ebf6127df6b74188
Merge: ec24d06 acb2915
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Tue Dec 3 15:40:15 2013 -0500
```

```
    fixed merge conflicts

commit ec24d06902b62a9cdac41696478d1cdf9248ea86
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Tue Dec 3 02:06:35 2013 -0500

    return index out of bound error instead of cryptic ocaml
error

commit c3836c447f87e2de442fa21428f0a5697c87378b
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Tue Dec 3 01:51:21 2013 -0500

    arrays pad out with types

commit 79618c44876b84d1e5dfef3af9d945a9a08f52b6
Merge: 6b782bb 03a8578
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Tue Dec 3 01:22:10 2013 -0500

    brought in master updates

commit 03a857842844c2e43593b2b83b0c4f8b7f210fcc
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Tue Dec 3 01:19:33 2013 -0500

    arrays scale to the given index. needs to get padded with
init values for the correct type

commit 6b782bb061428d9c78feacf95e034d953cb30fb5
Merge: 15fb29a b569f35
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Dec 2 23:29:57 2013 -0500

    Merge branch 'soundTypes' of
github.com:peterxu422/PLTCompiler into built-in-
functions

commit b569f3561a089c9df6b02e9f53418ed2fa18cb7e
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Dec 2 23:26:00 2013 -0500

    fixed assign test for new print function

commit 15fb29a2d3d3dd6a40125e7a1e51787cdaeaa944
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Dec 2 23:21:01 2013 -0500

    made getPitches work for our new Sound stuff

commit acb2915ca15b943e15368847c6dc32000f64f863
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Mon Dec 2 22:50:09 2013 -0500

    Fixed RETURN tests, all Statements implemented

commit 369d0711d4b8067fcbddb560f8bc6cea0c0129f8
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Mon Dec 2 22:43:50 2013 -0500

    FOR and WHILE statements implemented

commit e5f1bf6fefa147d75a3545bd069864be8d9657b5
Merge: 287885b e357433
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Dec 2 22:42:58 2013 -0500

    Merge branch 'soundTypes' of
github.com:peterxu422/PLTCompiler into built-in-
functions

commit e35743320566239ba2613a3e2062fb6a3eedcc52
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Dec 2 22:24:29 2013 -0500

    pitches are now more than just strings

commit 04330deebda7631ac67284807767083a1a1d3cbb
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Mon Dec 2 22:18:06 2013 -0500

    IF statements implemented

commit 28c1e2ccb168199a176019e2023f89f58b52582f
Merge: e483dae ca1f355
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Mon Dec 2 21:04:15 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler into nathan

commit ca1f3550da3e8c7ab8e8a27766d4bcf4ff872ba7
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Dec 2 20:15:53 2013 -0500

    removed tests for stuff we haven't tried to implement yet

commit 287885b688c2ecf7f6399d8d615825d80b206b81
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Dec 2 20:09:09 2013 -0500

    added getAmplitude and getDuration and their respective
tests

commit 2fedf39c35a71793964bcef777563afb3b023bcf
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Dec 2 20:02:18 2013 -0500

    added getPitch() for sounds and pitches

commit 1799146dd185bbb969feb6e79ef7af057bb2b5d8
Merge: f3f0091 9d8e893
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Dec 2 19:37:05 2013 -0500

    Merge branch 'soundTypes' of
github.com:peterxu422/PLTCompiler into soundTypes

commit f3f00917e42520ed9a7595e3392683d5c695dd56
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Dec 2 19:30:36 2013 -0500

    finished length() for arrays. updated test
```

```
commit 7d16aaa992a3f0f55cd5d014e0fd9c34381f14ca
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Dec 2 19:15:28 2013 -0500

    added length() for arrays

commit 977344218e4a5b20cda7c680b274318dea94f088
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Dec 2 18:50:57 2013 -0500

    fixed operator tests

commit 156a813b76291ad726b4b84281d154caa71599df
Merge: 4d586b9 697a503
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Dec 2 18:37:16 2013 -0500

    Merge branch 'master' of
github.com:peterxu422/PLTCompiler

commit 4d586b9e4f54fb941341d907f9ce943f16cc3f7b
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Dec 2 18:37:09 2013 -0500

    fixed test color bug

commit e483daed0618d58cd86a5604467b631d43f7b02b
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Mon Dec 2 00:53:38 2013 -0500

    Problems reading bool in interpreter.mll line 291

commit 9d8e893b39366e059614fd86fa47d658b6749d99
Author: Peter Xu <peterxu422@gmail.com>
Date:   Mon Dec 2 00:03:20 2013 -0500

    sounds changed from strings to tuple of string double
and int

commit 05f26da09ea1b9849f89c50179e7684c0b6fc1e3
Merge: ef4f4c6 697a503
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Sun Dec 1 23:23:14 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler into nathan

commit 697a503a347c12e360a6627fe73e0a4f5709c02a
Merge: 5f17a1b 4df7571
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Sun Dec 1 23:06:22 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 5f17a1b5a5a1f980f780ed50700c2a914cbab4f9
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Sun Dec 1 23:05:53 2013 -0500

    added functionality to Not and Neg operators

commit 4df75710fb326e60523467eac2384e397861886a
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Sun Dec 1 22:57:17 2013 -0500

    Added tester to master

commit 4c0f8962752d9b22ea5a6c722c1b808b49fafe14
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Sun Dec 1 21:47:20 2013 -0500

    added tests

commit d002d27607f28d5d6573b4040edd3347585e6dc3
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Sun Dec 1 21:42:18 2013 -0500

    removed files not needed added to gitignore

commit 14bc8357f9eb35f04ec2e9d7c0d64ccb145623eb
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Sun Dec 1 21:31:30 2013 -0500

    removed junk test dirs

commit 71588aeb6a64a1a6b2e8f4e05de683ca27a17822
Merge: fec3783 fe5aa0d
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Sun Dec 1 21:24:24 2013 -0500

    merged operators and array conflicts

commit ef4f4c6dffc67f08e6eaa2aeca7c5cfcb51bc03c
Merge: a80df1c 00bf513
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Sun Dec 1 21:15:04 2013 -0500

    Merge branch 'backend_array' of
https://github.com/peterxu422/PLTCompiler into nathan

commit a80df1c4e7bf9fa816a4ead4f78e82177f94023d
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Sun Dec 1 21:14:54 2013 -0500

    Makefile and testall placeholders

commit fe5aa0d0e25b8aa1b7cb13c2d39cd8ad6504870e
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Sun Dec 1 21:11:28 2013 -0500

    adding mixdown test

commit fec378337df11198a6da70ccda9758f29d378bb5
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Sun Dec 1 21:07:10 2013 -0500

    I undid these changes

commit 00bf513c041f6aead92cf080275e70b29787381c
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Sun Dec 1 21:04:05 2013 -0500

    pushing test-mixdown
```

```
commit 4f026c5de79e6321d24b84074e4d9e203bb7df1a
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Sun Dec 1 20:36:59 2013 -0500

    added jfugue-4.0.3.jar

commit 4b76c494a1b86904e5eef58d661284310597f24b
Merge: 8e0f710 0a90f38
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Sun Dec 1 19:56:29 2013 -0500

    Merge branch 'backend_array' of
https://github.com/peterxu422/PLTCompiler into nathan

commit 0a90f38097dbd97055459f52e74ef10d4f3d03d6
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Sun Dec 1 19:39:25 2013 -0500

    mixdown works calling it once, with one sound[]

commit 8e0f71002f9e423354564fc98dcc46798643eb47
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Sun Dec 1 18:40:54 2013 -0500

    Edited makefile to call testall.sh

commit 8d9d4396f227e45ab6162a5dd5e674d620365b53
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Sun Dec 1 18:37:35 2013 -0500

    Placeholder testall script

commit c09b163b3d5f47124502c7eafd752d5c216cdb5d
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Sun Dec 1 02:57:25 2013 -0500

    Run tests using 'Make test'

commit d3355f8e4127da0d4508d5ef5bde672bdda97ba1
Merge: e06afb9 fc8f94f
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Sun Dec 1 01:53:51 2013 -0500

    Merge branch 'backend_array' of
https://github.com/peterxu422/PLTCompiler into nathan

commit e06afb93feb13266effc04fa02bdc909760a18c4
Author: nhayes-roth <nathan.hayesroth@gmail.com>
Date:   Sun Dec 1 01:34:59 2013 -0500

    Fixed errors in Makefile

commit dbc8d4a722c3bd5ad6e46600fd38865c8881762f
Merge: 55ca91c fc8f94f
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Sat Nov 30 17:14:05 2013 -0500

    Merge branch 'backend_array' of
https://github.com/peterxu422/PLTCompiler into backend-
mixdown

commit 55ca91c1fc514c7dbb9a74765408d1983dcc283d
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Fri Nov 29 20:09:59 2013 -0500

    Added Binop operators for doubles, floats, and booleans.
Tested using backend\ tests/testOperators.llb

commit fc8f94f9d3105e9354cefdecbca915ab5af15340
Author: Peter Xu <peterxu422@gmail.com>
Date:   Fri Nov 29 04:57:32 2013 -0500

    implemented assign value to array index

commit c0f19958c8a9ddb34900a1227cf003f088f5d6c1
Author: Peter Xu <peterxu422@gmail.com>
Date:   Fri Nov 29 03:22:05 2013 -0500

    array access implemented

commit d179ea5b91175c561a977a2c3a1cfcc6aa335f6e
Author: Peter Xu <peterxu422@gmail.com>
Date:   Thu Nov 28 18:13:12 2013 -0500

    readded test-if.llb

commit 948d593623b401ebc4a09054785dafcc21a2a76c
Author: Peter Xu <peterxu422@gmail.com>
Date:   Thu Nov 28 18:03:42 2013 -0500

    implemented variable assignments

commit cddef5d0ba693251b9026adc6c4ac81fba8e354c
Merge: 141e4b4 ae54c8e
Author: Peter Xu <peterxu422@gmail.com>
Date:   Tue Nov 26 14:46:21 2013 -0500

    merged backend_array with front end. fixed merge
conflicts

commit ae54c8e1a7eef43d29194c7f394c365a99dedc1e
Author: Peter Xu <peterxu422@gmail.com>
Date:   Tue Nov 26 14:44:01 2013 -0500

    moved type literal evaluation to be lexed before
identifiers in scanner. This is the right order of evaluation,
do not change it

commit 141e4b486408e1f1324d4fa09db3b050335d756a
Author: Peter Xu <peterxu422@gmail.com>
Date:   Tue Nov 26 14:41:28 2013 -0500

    printing for array pitch and sound literals implemented

commit 39e2d58921cc5d8c6f7d00b99e46110ff3edd941
Author: Peter Xu <peterxu422@gmail.com>
Date:   Tue Nov 26 11:35:46 2013 -0500

    clean up frontend tests directory

commit 8b7f11973c0c0929bb02f4e94e47149e38ba022c
Merge: 84d8a04 3eadc25
Author: Peter Xu <peterxu422@gmail.com>
```

```
Date:   Tue Nov 26 11:30:22 2013 -0500

    Merge branch 'frontend' into backend_array

commit 3eadc254c8863f97dd14c5aef3d995b1e026e5a1
Author: Peter Xu <peterxu422@gmail.com>
Date:   Tue Nov 26 11:21:26 2013 -0500

    modified test files

commit a3e6e118875fd099c805764f8427f289713ed8e8
Author: Peter Xu <peterxu422@gmail.com>
Date:   Tue Nov 26 11:20:09 2013 -0500

    added prettyprint for array access

commit ff02ebf228ebe261aae65144757046e0f3a2958d
Author: Peter Xu <peterxu422@gmail.com>
Date:   Tue Nov 26 10:42:17 2013 -0500

    renamed front end test fiels

commit 84d8a0400e7996efbee25276ee609b8c66da360e
Merge: 5c2cce0 9966dd2
Author: Peter Xu <peterxu422@gmail.com>
Date:   Tue Nov 26 03:54:51 2013 -0500

    Merge branch 'frontend'

commit 9966dd2d0c5aee04eff5c5f8ff24304b4680f85d
Author: Peter Xu <peterxu422@gmail.com>
Date:   Tue Nov 26 03:25:33 2013 -0500

    added pretty print for array

commit 5c2cce0a6a400ba34bc9ccd82bd07b484e63fac2
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Sun Nov 24 21:04:36 2013 -0500

    fixed comment bug

commit d8033a85aa989b4b65244676400358fdf076312a
Merge: 6313a04 8d0e71f
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Sun Nov 24 20:53:23 2013 -0500

    merged master to backend

commit 6313a043caa2e97b80c1d21a2814735998256ffa
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Sun Nov 24 20:51:14 2013 -0500

    no errors, not much is here though

commit 8d0e71f485ef4f0738453bef30ac5f47d7b7555f
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sun Nov 24 20:15:22 2013 -0500

    added make.sh

commit df9e419021cd796e4f2f3d76c0ddccf7cb4c0418
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Sun Nov 24 20:12:05 2013 -0500

    needed an s

commit 3ca81bd99655d99e43b7bff8c3eba1a199769acc
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sat Nov 23 03:28:11 2013 -0500

    made Makefile more formal

commit 7d91cd337fa0a6edcd0cdf0a4681e1d55e1a4c9e
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sat Nov 23 03:10:01 2013 -0500

    Created a proper makefile

commit ef4179f24dc78a66dd94ac71ad7f331a3983c4ee
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sat Nov 23 03:09:41 2013 -0500

    added statement test programs in lullabyte for interpreter
and expected output

commit 065fb31e9418fdf3cd45d2494e35334c5a40aa15
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sat Nov 23 03:08:29 2013 -0500

    added interpretation for return while and loop and (expr)

commit 7b8b89c1b2ab9195d4b84aa2e54c2b67d4afc51c
Merge: 2f17668 ea47dd0
Author: Peter Xu <peterxu422@gmail.com>
Date:   Fri Nov 22 18:54:03 2013 -0500

    fixed conflicts from merging with operators in scanner
and parser

commit 2f17668f5d3bc71cadcb5a9d8d8c69ca99d2d930
Author: Peter Xu <peterxu422@gmail.com>
Date:   Fri Nov 22 18:50:03 2013 -0500

    added test cases for front end

commit 17f375949e7cc4979f62d1d531bdbfb63119d7f8
Author: Peter Xu <peterxu422@gmail.com>
Date:   Fri Nov 22 18:49:11 2013 -0500

    added prettyprint for if else statements

commit ea47dd04c6565c9d498dc205d7287c5ee4243f7c
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Thu Nov 21 21:35:11 2013 -0500

    added negative operator

commit 4cb15c0d81a5f9bbf829dcee7309fec79a2afc30
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Thu Nov 21 21:16:57 2013 -0500

    added and tested expressions for ast

commit 2e14003598128c5f06660caebf347fe18485363d
```

```
Author: Peter Xu <peterxu422@gmail.com>
Date:   Thu Nov 21 20:03:57 2013 -0500

    pretty print sound lits

commit 345506435f554fedb52939669b9981aca8d009c4
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Nov 21 18:57:08 2013 -0500

    actuals work, but there's one thing that doesn't work, bad commit but I had to

commit 349bb949ad9efdfa0b6892ace79f0d1201ca1e0b
Merge: e53fa7f afd07bd
Author: Peter Xu <peterxu422@gmail.com>
Date:   Thu Nov 21 18:16:12 2013 -0500

    Merge branch 'frontend' of https://github.com/peterxu422/PLTCompiler into frontend

commit e53fa7f5521daaf92987387219e021504cb30d4f
Author: Peter Xu <peterxu422@gmail.com>
Date:   Thu Nov 21 18:15:55 2013 -0500

    incorporating sounds, merging andrews push to frontend

commit afd07bd420f134827ce5f367b26b7dcd6771de8f
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Nov 21 18:14:36 2013 -0500

    added paramaters to frontend

commit ed36e202382a986aa93dd23b00211daa9c4c8f44
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Nov 21 17:30:12 2013 -0500

    Update .gitignore

commit bb745bd70ce8feb93ed1ed4f285f5ec010e3c81e
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Nov 21 17:29:13 2013 -0500

    Delete andrewscompiler.ml

commit b134ade6d47f0bbf68f44a6eb7d1fa02c085c3ae
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Nov 21 17:23:09 2013 -0500

    backend: call is being called on expressions

commit 662cbac1193dc05e4b41efe43fa49dcf00c4d7c0
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Nov 21 16:36:52 2013 -0500

    frontend: fdecls an rtype and local variable declarations

commit 504eece303e4de67316a208563728327f1c84fdc
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Nov 21 16:12:46 2013 -0500

    testBackend fix

commit 2cea7e80dc3c1948b7e6718c674b72fa360a7ed8
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Nov 21 16:05:49 2013 -0500

    fixed the interpret test

commit cd9e76741d8a5718b06860d40fda2902bbfd7923
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Nov 21 16:02:37 2013 -0500

    Added assign token

commit 1d5424658fbbb209a4387090dba483c79f13c807
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Thu Nov 21 15:43:55 2013 -0500

    added front-end test, do ./testFrontend.sh

commit e3632f65382fa4437f1d0d97857a5c20808b88fe
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Wed Nov 20 23:07:12 2013 -0500

    print for the boys

commit 1b3c3e2eb989d59ea1db2670fa5b1c2ad27a057d
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Wed Nov 20 20:53:42 2013 -0500

    added frontend test file. ./interpret < frontEndText.llb

commit 5078d4a16ee2d485b4078d23cc661b0805ef3e17
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Tue Nov 19 21:45:44 2013 -0500

    arrays are recognized

commit f0e6c5cd7d301b7a7ee07794569e2143a8eac1cf
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Tue Nov 19 21:14:00 2013 -0500

    pitches are now a thing

commit 7fd346af9fe2147eef7d1f2699abc3c3473ebe35
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Tue Nov 19 21:02:17 2013 -0500

    added boolean datatype

commit 132f50d4c0388734440e8e03c0fba2744480b9ba
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Tue Nov 19 20:53:22 2013 -0500

    double keyword is recognized and so are doubles

commit 4cc5fd6583d566a8cafdf506a77d1b504e620296
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Tue Nov 19 20:37:29 2013 -0500

    started types and vdecl

commit f18c408d8b036ad46f4e72800eaa026f3daa36ac
Author: Peter Xu <peterxu422@gmail.com>
```

```
Date:   Sun Nov 17 23:43:56 2013 -0500

    make2

commit b52c3bdc3816686115270278c2245d9585f56023
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sun Nov 17 23:21:03 2013 -0500

    updated parser

commit 935d69b05939c02b4f1b763768472df597db2b80
Merge: 6f17ce7 38b9d4a
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sun Nov 17 23:10:14 2013 -0500

    making interpreter

commit 6f17ce7a82da923eceb33ba11fc19f46d5f5ddd1
Merge: 5aad93e caa08d7
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Mon Nov 11 22:44:17 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 5aad93e3e0c773e954495dfcc1b8d0f8ddd53d16
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Mon Nov 11 22:43:56 2013 -0500

    fixed scanner with tests

commit caa08d77b8e1767ab5f3092cc3b2f21a26ea8b87
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Nov 11 22:41:45 2013 -0500

    added gitignore

commit 38b9d4aa5b1e6c76b28cf4380d58f0ef1182ca05
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Mon Nov 11 22:37:22 2013 -0500

    recognizes print

commit 36abd0a676cf41ab6df445706b6d0263a6bbb18e
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Mon Nov 11 21:45:18 2013 -0500

    make.sh

commit f56370c3d7cc73e176718d4f10cfdce06a313b82
Merge: 4a9f6eb 9dc158c
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Mon Nov 11 15:30:52 2013 -0500

    created mixDown to the best of my ability

commit 4a9f6eb5de08c9b6859c725adbf71058e4e6f8d0
Author: Louis Croce <ljc2154@columbia.edu>
Date:   Mon Nov 11 15:29:39 2013 -0500

    Created mixDown.ml to the best of my ability.  Please
look at comments.

commit 9dc158c49e447672f7781815ab0fe8abeaa3228c
Author: Peter Xu <peterxu422@gmail.com>
Date:   Mon Nov 11 14:20:50 2013 -0500

    created ast, updated scanner and parser

commit 8b63856198c14d3bf91ef3393b740cdbf483b39e
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Mon Nov 4 00:10:56 2013 -0500

    push for clarity in tokensTest.llb

commit 47f895f76f5596dcead0fe393da55018ca0868f0
Merge: 05e9f4f e6d97b3
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Mon Nov 4 00:08:50 2013 -0500

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 05e9f4fbe671cc2388d1e4d165996225f906330d
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Mon Nov 4 00:08:01 2013 -0500

    started testing suite with scanner tests

commit e6d97b32ebc9dcac21f34332336c9019ab4decaa
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Sun Nov 3 21:56:12 2013 -0500

    added bytecode translator javafile

commit 94f2edca7850e9b5ee678757473648a407fa96fd
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Sun Oct 27 20:29:51 2013 -0400

    Delete test

commit a8883151e77767c0ef50b52db87eea71b52779ac
Author: Andrew Langdon <andrewlngdn@gmail.com>
Date:   Sun Oct 27 20:29:36 2013 -0400

    test push

commit a3c4dfb9561b87808beddfb0609326b42e68409b
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Sat Oct 26 23:05:57 2013 -0400

    added test file for scanner

commit 6fb555b9a0803c99b7994def68b0743feba4a85f
Merge: 0c4d8c3 98d324f
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Sat Oct 26 22:19:51 2013 -0400

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit 0c4d8c35eaa5f2bba39b6c46380b0c063e437fec
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Sat Oct 26 22:19:05 2013 -0400
```

```
    added sound token

commit 98d324f253c5c5ca325031c89831be7c7fd27dc5
Merge: 8d7254b 4845142
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sat Oct 26 22:15:55 2013 -0400

    merged statements and expressions

commit 8d7254b3c0555e9a40a358d37be648fb6d2c2fed
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sat Oct 26 22:08:33 2013 -0400

    parser statements completed without SR errors

commit 484514287b6978a6dc29357103eca5b98a614d4e
Author: tehapple <stanley0856@gmail.com>
Date:   Sat Oct 26 21:47:51 2013 -0400

    finished expr

commit e06cc4daad6bf38f2a6642d180ec120dd3e7386f
Merge: ed0a740 e823bd7
Author: Stanley <tehapple@ubuntu.(none)>
Date:   Sat Oct 26 21:05:09 2013 -0400

    fixed merge

commit ed0a74052cab77134e3b61c29df7ab8e7e2c3be4
Author: Stanley <tehapple@ubuntu.(none)>
Date:   Sat Oct 26 20:59:27 2013 -0400

    started expr

commit e823bd76cbb8bd0a8f8c15df637edeefd3b8a8fc
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sat Oct 26 20:58:54 2013 -0400

    statements rules added

commit dd8f909c472f06b91a471a16900577063a39af4c
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sat Oct 26 20:12:20 2013 -0400

    parser first iteration

commit a3e34302c2af3e8b07d20ecb1f42c18d83439e94
Merge: 67c1f01 ec4f6d6
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sat Oct 26 19:32:29 2013 -0400

    Merge branch 'master' of
https://github.com/peterxu422/PLTCompiler

commit ec4f6d675544d2dfe88e1a23ef720e2d681c34e9
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Sat Oct 26 17:25:20 2013 -0400

    added percent or and // pitch const

commit 67c1f0167684a63226571442f50077869130889a
Author: Peter Xu <peterxu422@gmail.com>
Date:   Sat Oct 26 15:50:07 2013 -0400

    delete asdf add eeeee

commit 8b028191b9b00ed30e01cc6be4d015b03be1cc93
Author: Ben Nappier <ben2113@columbia.edu>
Date:   Sat Oct 26 15:37:08 2013 -0400

    added the file

commit 231cf3ec1d0d1dbdd21d533b4aff983fb4ee7d20
Author: Peter Xu <peter@ubuntu.(none)>
Date:   Sat Oct 26 13:26:35 2013 -0400

    parser and scanner initial

commit 80735cbe469f8abdc19f200c50bf5ae923eb325d
Author: peterxu422 <peterxu422@gmail.com>
Date:   Sun Sep 22 17:13:37 2013 -0700

    Initial commit
```