PLT Final Report
Analog Additive Synthesis Language (AASL):

A programming language for the creation of original sounds

Vaishnav Janardhan, Rob Katz, Carlos Rene Perez, Albert Tsai

May 7th, 2007

Table of Contents

Chapter 1

Introduction

Abstract

The purpose of AASL is to simplify the creation of original sounds by allowing users to easily generate and combine the fundamental elements of sound composition - oscillators, mixers, and envelopes - without needing to create their own data structures and methods to do so. In short, the goal is to present those elements as atomic data types which are then manipulated by intuitive operators or functions.

A Brief History

A synthesizer refers to a device that can create electronic signals for use as sounds. The history of such devices begins with the Telharmonium in 1876, an electromechanical instrument. Its inventor, Thaddeus Cahill, wanted to amplify electrical signals to produce audible noise. To do so, he created tonewheels - wheels with raised bumps. These tonewheels were held close to magnetic coils, and the tonewheels were spun by an electric motor. When the bumps were close to the coil, electricity was generated through induction. In the distance between the bumps, little electricity was generated. This produced an alternating current, which, when piped to a telephone receiver, could be heard as a steady pitch. By altering the number of bumps on the tonewheel, the angular velocity, and the distance of the magnetic coil to the tonewheel, a wide range of pitches could be created. And by combining multiple tone wheels, the Telharmonium became polyphonic. Cahill designed the machine so that the tonewheels could be controlled by a combination of a keyboard and foot pedals.

Unfortunately, the volume of the sound depended on the size of the generator. Eventually, Cahill's Telharmoniums ended up several tons in size.

The first programmable electronic synthesizer arrived in 1957 - the RCA Mark II Sound Synthesizer by Vladimir Ussachevsky and Peter Mauzey. It was programmed with punch paper roll which specified the frequency, octave, envelope, timbre, and volume for 24 variable vacuum tube oscillators. The Mark II would read in the paper tape, and output sound to a synchronized shellac record lathe next to the machine. Sophisticated mathematical/musical operations were available, such as high and low pass filtering, noise, glissando, tremelo, and patchable resonance and attenuation sections.

Alas, the analog circuitry often had to be rewired before playing a program. And because it couldn't be played in real time, and little effort was made to make the programming accessible to layman composers, the Mark II failed to achieve mass popularity.

Nevertheless, not just technologically, but musically, the Mark II was significant because it granted composers the freedom to write music using rhythms and tempos that were not practical or possible with acoustic instruments. This precision was seen as a mark of aesthetic progress, and contributed to the increased awareness of electronic music as a viable new art form.

The RCA Mark II can currently be seen at the Columbia-Princeton Electronic Music Centre.

Analog synthesizers really came of age when Robert Moog introduced his synthesizers in 1964. The invention of the transistor had revolutionized computing machines by reducing power and size requirements, and synthesizers followed suit. With transistors replacing the vacuum tubes, a keyboard interface, and voltage controlled oscillators and envelope generators operated by knobs, the Moog synthesizers brought the capabilities of the Mark II (and more) to the average composer. Notably, Moog's synthesizers were modular in design - the output of one oscillator could be connected to the frequency/amplitude of another oscillator. At last, the world had a synthesizer whose weight wasn't measured in tons! It fueled an explosion of music incorporating (or wholly composed with) synthesized sounds.

This history is one of engineers and musicians trying to create novel new sounds and compositions with the newest technological developments. Modern synthesizers are basically electronic instruments, easily learned and played by anyone with familiarity to musical instruments in general. However, we believe it would be fruitful to explore a niche area, where the fields of computer science and music converge. This class in compilers gives us the opportunity to create a programming language orientated to the synthesis of new sounds. With it, precise sounds can be made by specifying frequencies and amplitudes with integers, instead of knobs, and mathematical relationships expressed in music. What does Pythagorean's Theorem sound like? Well, if one oscillator was at frequency a^2, another oscillator at frequency b^2, and a third at (a+b)^2, you could find out...or you could create compositions that were not humanly playable...or you could create normal pieces as well, of course. These are but toy examples, but they do express the possibilities with our language.

We can see common musical components in all synthesizers. Oscillators (beginning with Cahill's tonewheels) produce electronic signals, which are described by three attributes - a waveform, a frequency, and amplitude. These synthesized signals are combined with mixers to produce a final sound - in modern language; this process is called additive synthesis (as compared to subtractive synthesis). These elements correspond naturally with the data structures in the sound composition language we have produced.

Oscillators: An oscillator is a waveform (sine, square, saw, etc), with an associated frequency and amplitude. Osc (form,freq,amp)
Oscillators can be controlled by envelopes, which are non-periodic functions over a set period of time, with value from 0-1. The envelope can be used to control the relative strength of an oscillator's component.
Oscillator elements can also be controlled by other oscillators!
Mixers combine/sum input oscillators.

Bibliography

----
[1] http://www.obsolete.com/120_years/
[2] http://www.synthmuseum.com/

Chapter 2

Language Tutorial

Running the compiler

The compiler is executed by calling AaslMain main function with the file name as a parameter or with its absolute path name. Then according to the OUTPUT = "helloworld" statement of your program, the compiler will create two files: helloworld.java and helloworld.wav. The former being the intermediary code generation and the latter the actual output wave file the is to be played.

Example Program: HelloWorld

The following program will output a simple sine wave at a frequency of 440Hz and at amplitude of 1.

```
-------------------------------------------
start header
OUTPUT = "helloworld"
TONELENGTH = 3 // length in seconds
end header

start func main
    start def
        osc o1="SINE";
    end def

    start connect
        osc o1(440,1.0);
    end connect

    OUTPUT = o1;
end func main
--------------------------------------------
```

Program Sections

The most important sections are the "Header" and the "main" function, they are mandatory. They specify the behavior and characteristics of the output wave format. All sections are delimitated by "start [sec_name]" and "end [sec_name]". There are a total of five section statements in the AASLanguage. Some types of sections can be nested within others.

Here we see the two primary sections of an AASL program, "header" and "main". "def" and "connect" will be discussed afterwards. Header and main are required in every AASL program. The header section defines the output of our program - in this case, a wav file named "helloworld" that is 3 seconds long. The wav file will be placed in the same directory that HelloWorld.aasl resides. The second section is a special function, the main function. Main is looked for in every AASL program, and it is where execution of the program begins.

Main Function

Within the main function (and all functions) there are three sections, "def", "connect", and OUTPUT (an implied section). The historical practice of synthesizers has been to, first creating elements (oscillators), and then connecting them (see Moog). Therefore an AASL program emulates this practice with the definition and connection sections.

In the definition section, we define just one element, an oscillator with a sine waveform. Three other waveforms are possible: "SQUARE", "SAW", and "REVSAW".

In the connection section, we connect the oscillator to a constant frequency of 440Hz, and to amplitude of 1.0. In our language, amplitudes are defined with relative values, from 0.0 to 1.0. These values are factors multiplied against the sound's maximum amplitude. Simply, <amplitude value> * <maximum amplitude>. So in this example we are outputting the maximum amplitude. "Connecting" to constant values seems a little awkward here - this statement seems more like a definition. But later on, we'll see an oscillator's frequency and amplitudes controlled by other oscillators, which with physical synthesizers required a literal connection of one oscillator module to another. So that practice is the motivation for the syntax.

The OUTPUT statement simply identifies which oscillator will be saved as the "helloworld" file, as previously declared in the header section. Here we output our only oscillator o1.

Example 2

Let us introduce some more elements of the AASL language - Oscillator Banks, Envelopes, and Mixers.

```
-------------------------------------------
start header
OUTPUT = "osc2osc"
TONELENGTH = 12
end header

start func main
    start def
        oscbank oscb1 = 3;
        oscb1[0]="SQUARE";
        oscb1[1]="SINE";
        oscb1[2]="SAW";

        env e1 = { (0.0,0.3) (0.2,0.0) (0.4,1.0) (0.8,0.8) (1.0,0.0) };

        mixer s1;
    end def
    start connect
        oscb1[0](oscb1[1]@1000,e1);
        oscb1[1](oscb1[2]@3,1.0);
        oscb1[2](440,1.0);

        s1 = oscb1[0] + 2*oscb1[2];
    end connect

    OUTPUT = s1;
end func main
-------------------------------------------
```

We can see the keywords for the three elements: "oscbank", "env", and "mixer". We also see a strange new beast used as an argument in the oscillator connection functions: "oscb1[1]@1000".

Oscillator Banks

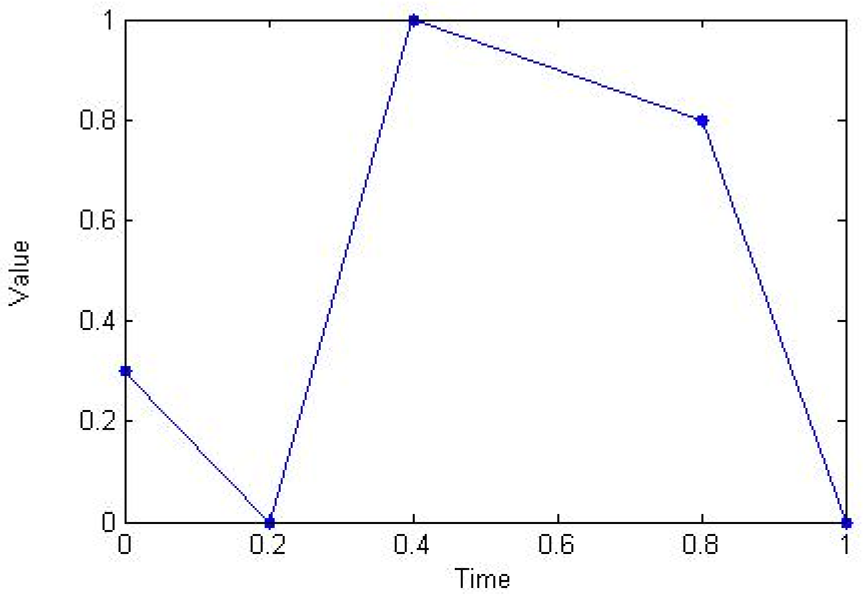
An oscillator bank is simply a collection of oscillators, stored as an array and indexed by integer. They exist as an easy way to group oscillators, and to make it easy to operate on them in conjunction with control statements, which we'll see in the next section. Wherever an oscillator can be present in an AASL program, so too can be an oscillator bank with an index (i.e., oscb1[1]). You first declare the oscillator bank in the definition section, with an argument for how many oscillators you want to store in it. Then you need to define a waveform for each (as you would for normal oscillators). In the connection section, they are connected to frequencies and amplitudes as before.

Envelopes

Envelopes are <x,y> pairs that describe a piecewise function over the length of the whole song. x and y are fractions that can take values from 0.0 to 1.0, inclusive. Specifically, x represents the point in time that the specified fraction of the sound has been reached, while y is a fraction to be applied to an oscillator's amplitude or frequency (depending on the envelope's use in the connection section). Additionally, the envelope doesn't describe a point function, but a continuous function that linearly connects all of the points specified in the envelope.

Put another way, the graph for
env e1 = { (0.0,0.3) (0.2,0.0) (0.4,1.0) (0.8,0.8) (1.0,0.0) };

Looks like:



When this envelope is attached to an oscillator's amplitude, as in the example, oscb1[0]'s amplitude at the beginning of the song will be 3/10th's of the maximum. From the beginning of the song until 2/10th's of the song, the amplitude will smoothly and linearly decrease to 0, and from 2/10th's to 4/10th's, will increase to the maximum. You can quickly see that envelopes are useful for fade-in and fade-out effects when attached to oscillator amplitudes.

This capability would be useful to modulate frequencies as well. E.g., if you wanted to create a siren noise, you would want to modulate back and forth from a low frequency to a high frequency. You can't just attach an envelope we've defined to the frequency of an oscillator, as we did for the amplitude - you need something to multiply the envelope values against. For amplitudes, there was an implied argument; the sound's maximum amplitude. To connect to a frequency, you need to specify a central frequency. Hence the argument

e1@1000

will modulate the frequency of 1000Hz over the course of the oscillator, as specified by the envelope. You aren't limited to connecting envelopes to oscillator frequencies either. Notice in our sample code that we connected *another* oscillator, modulating a 1000Hz signal, to oscb1[0].

The main difference between oscillators and envelopes is that oscillators are periodic functions, whereas envelopes are functions that span the length of the whole sound.

Mixers

You simply hand a mixer a bunch of oscillators to add (superimpose) together. In addition, you can use basic mathematical expressions to modify the amplitude of an oscillator before it gets mixed.

S1 = 0.4*o1 + 0.6*o2;

Here, mixer S1 superimposes oscillator o1 and o2 together. However, before they are added, o1 is filtered so that only a fourth of the amplitude is expressed. o2's amplitude is filtered to a sixth of its strength. Note that 0.4*o1 does not set o1's amplitude to a constant 4/10ths for its entire length.

Mixers mix whole oscillators – they ignore segments. Don't worry, segments are explained next.

Example 3

This example features a user-defined function that takes an integer and creates two oscillators who's frequency values depend on the argument. The function is used like a single oscillator in assignment of mixer s1. Furthermore, we are using the if control structure.

```
--------------------------------------------
start header
    OUTPUT = "userDefFunction1"
    TONELENGTH = 3
    SEGMENTS = 2
end header

//function definition...takes one integer argument
start func udefFunc(int x)
    start def
        osc oUdef1="SINE";
        osc oUdef2="SINE";
        mixer s1;
    end def

    start connect
        oUdef1(x, 0.75);
        oUdef2(0.75*x, 0.75);

        s1 = oUdef1 + oUdef2;
    end connect

    OUTPUT = s1;
end func udefFunc

start func main
    start def
        int x=2;
        int y=1;
        int z=x+y;
        oscbank oscb1 = 2;
        if(SEG==1){
            oscb1[0]="SINE";
        }
        else{
            oscb1[0]="SAW";
        }
        oscb1[1]="SQUARE";

        osc o1="REVSAW";
        mixer s1;
    end def

    start connect
        oscb1[0](z,y);
        oscb1[1](oscb1[0]@440,1.0);
        o1(440,1);

        s1 = udefFunc{1000} + 2*o1 + 4*oscb1[1];
    end connect

    OUTPUT = s1;
end func main
------------------------------------------
```

This is our last, most complex example. Here we see three new elements: Segments, if/for loops, and user defined functions.

Segments

Segments are declared in the header section. They divide the sound into *n* segments, with each segment at equal length of TONELENGTH/SEGMENTS. With the keyword SEG, you can use them within control statements to efficiently define/connect different oscillators to different segments of the sound.

if/for loops

if/for loops behave like they do in traditional imperative languages like C and Java. Execute the body of the statement until the condition is satisfied. Note that within definition sections, if/for statement bodies may only contain definition statements, and within connection sections, if/for statements may only contain connection sections.

If statement reveals the utility of the SEG keyword. The most important feature of the IF control statement is to set the current Segment Array in that specific code block. In other words, the sound file is seperated into N segments (defined in the header section), each data type except the mixer has a different oscillator/envelopes for each segment. A Segment Array being simply a bitmap array, would allow all statements in the IF control block to be evaluated depending on SEG==x to ONLY that SEGMENT.

If there is an OR statement such as "if (SEG==1||SEG==2)" then for that control block the bitmap array would be set true for segments one and two. On the other hand, if there is an AND statement such as "if (SEG==1 && SEG==2)" that statement represents a situation that is physically impossible for a sequential wave function and thus it throws an error! The only other AND statement that is allowed is "if(x==1&&SEG==2)" or "if(x==1&&x==2)" boolean expression.

User Defined Functions

For user-defined functions, the compiler stores references to every function that it finds before the MAIN function. It stores the function name and parameters list type, so as to determine which function the user want to call (polymorphism). Once in the MAIN function the compiler sees a function call, looks up the argument types passed to the function call and searches for a function prototype that matches the specified function call parameters. If more than one function matches, an error is reported, otherwise the compiler creates all the data types and connection statements needed to handle the function execution in the code generator. Thus, user-defined functions behave like any other MAIN function, they only differ in name and in function parameters.

Chapter 3

Language Manual

AASL Language Reference Manual

Group Members:
Vaishnav Janardhan, Rob Katz, Carlos Rene Perez, Albert Tsai

1. Introduction:

The purpose of AASL is to simplify the creation of original sounds by allowing users to easily generate and combine the fundamental elements of sound composition - oscillators, mixers, and envelopes - without needing to create their own data structures and methods to do so. In short, the goal is to present those elements as atomic data types which are then manipulated by intuitive operators or functions.

AASL programs require a basic structure. They are divided into the following sections:

Header - Specifies the configuration of the output sound file

User-defined function(s) - Helper functions that the user sees fit to create.
Declaration section
Connection section

Main function - The first function to be executed
Declaration section
Connection section

The composition of these sections will be described in more detail later on.

1.1 Document Conventions

The word "elements" in this document will hereafter refer to "oscillators, oscillator banks, mixers, and envelopes."
Because AASL programs are structured, there are aspects of the language (statements, for one) which are only applicable to certain sections of a valid AASL program. These will be identified with tags such as [Declaration Section].

2. **Lexical Conventions**

A character stream first must be transformed into the valid "words" of a language; alternately stated, the atomic units of a valid AASL program. These units are called tokens.

2.1 Tokens

There are seven classes of tokens: identifiers, keywords, string literals, constants, operators, and grouping tokens (parentheses and curly braces). "White space", which includes spaces, tabs, newlines, and comments, is ignored except in as much as they separate tokens.

2.2 Comments

The language allows for both single-line and multi-line comments. Single line comments are initiated with a double-slash (//) and run until the next newline. Multi-line comments are initiated with a slash-star (/*) and terminate with the first occurrence of star-slash(*/).

2.3 Identifiers

Identifiers are sequences of letters, digits and underscores that begin with a letter. The language is case-sensitive, so 'word' and 'Word' represent two different identifiers. The maximum length of an identifier is not defined by the language.

2.4 Keywords

The following are reserved keywords and may not be used as identifiers.

mixer	start	end	header	func	connect	env	osc
def	oscbank	if	else	for	int	float	TONELENGTH
SINE	SQUARE	SAW	REVSAW	OUTPUT	SEGMENTS	SEG	

Keywords can be used for the following reasons:

- To define or create a data type (mixer, env, osc, oscbank, int, float)
- To declare a waveform (SINE, SQUARE, SAW, REVSAW)
- To define a statement or to be used as part of a statement (if, else, for)
- To define a function (func)
- To define a block (start, end, header, connect)
- To define language output parameters (TONELENGTH, OUTPUT, SEGMENTS, SEG)

2.5 String Literals

String literals are surround by double quotes, and are context limited to the header section as the assignment r-value of the OUTPUT keyword.

2.6 Constants

There are two types of constants allowed in AASL, integer-constants and float-constants. Both are numeric constants.

2.6.1 Integers
Integers are decimal whole numbers represented as a sequence of digits.
2.6.2 Floats
Floats in AASL are represented as a sequence of numbers with a single period. There is no support for exponent notation.

2.7 Operators
By and large, all of the following have the same meaning as one would expect. The period operator is similar to that found in most object oriented languages. In AASL, it is used in the context of an oscillator bank, which represents a group of related oscillators, to distinguish one from the other. For example, in a bank of called oscbank1 with two oscillators, they would be refered to as oscbank1[1] and oscbank1[2]
The individual elements of oscbank are referenced as it is done in C-Programming language. The i-th element is accessed by the operation <oscbank-obj>[i-1]

+ - * / % = ! < > <= >= == && || ++ -- [ ]

1.8 Other Tokens
( ) { }

Parenthesis are used in the following contexts.
- In the definition of an envelope, surrounding each time/amplitude pair.
- In connection expressions.
- Surrounding the conditions of an if or for statement.

Braces are used in the following contexts.
- Surrounding the argument list in a function declaration or call to a function, following the function name.
- To contain the bodies of code following conditionals.
- To contain a complete set of time/amplitude pairs in the definition of an envelope.

3. Meaning of Identifiers
Identifiers, or names, refer to functions and objects. Functions are the subroutines of a larger AASL program; they exist for the user's convenience in organizing and programming AASL code and will be discussed in Section 7, Functions. Objects are named regions of storage containing instances of primitive data types such as integers and floats, or instances of the four fundamental data types of an AASL program: oscillators, oscillator banks, envelopes, and mixers.

3.1 Data types
The standard integers and floats are available in AASL programs. Additionally:

1. Oscillators - Oscillators output a signal in the form of an array of audio samples (44.1k per second). The size of this array is dependent on the declared length of the tone

(44,100 * (length in seconds) elements). An oscillator has three defining characteristics, namely, a wave shape, a frequency, and an amplitude. The frequency and amplitude may be constant or controlled by another element.

2. Oscillator Banks - Oscillator banks are simply groups of oscillators. The intention is that the oscillators are somehow related and will often have their characteristics defined in some sort of conditional loop. Individual oscillators within an oscillator bank are referenced in C array notation. For example, given an oscillator bank name ob1 with three oscillators, the individual oscillators may be referenced as ob1[1], ob1[2], and ob1[3].

3. Envelopes - Envelopes are representative of a Time v. Amplitude Graph over the entire tone length. Each member of a pair is a float between 0.0 and 1.0, inclusive. Envelopes are "connected" to an <u>oscillators</u> *frequency* or *amplitude*, which then tracks the graph. For example:

3a. We have an envelope with pairs (0.1, 0.0) , (0.5, 1.0) , (0.9, 0.5) connected to the ***amplitude input*** of an oscillator. One-tenth of the way through the tone, the oscillators amplitude will be zero. Half way through the tone, it will be at full volume. Nine-tenths of the way through the tone it will be back down to half amplitude.

If no amplitude value is provided for time zero, as in our example, we assume a starting volume of zero with a linear increase to the first defined value.

In our example, this value is 0.0, so the volume stays at zero for the first tenth of the tone. Between times 0.1 and 0.5, there is a linear increase in volume from 0.0 to 1.0. After time 0.9, the volume holds steady at 0.5.

3b. If an envelope is attached to the ***frequency input*** of an oscillator, a central frequency *must* be supplied. This will be the oscillators output frequency when the envelope's value is 0.5. Similarly, if an oscillator is connected to another oscillator's frequency input, a central frequency must be provided.

4. Mixers - A mixer takes the outputs from multiple oscillators and sums them, generating a new output. The input oscillators may be mixed in unequal amounts, but the output of a mixer is never greater than the maximum output of a single oscillator. A mixer must have at least two inputs.

4. Expressions

Expressions are combinations of values, objects and operators that are evaluated according to the particular rules of **precedence** and **association** to return a value. Common locations of expressions in an AASL program are in conditional blocks, envelope definitions, indexes of oscillator banks, and connection statements.

The precedence of expression operators is organized in the following table as highest to lowest, top to bottom. Within each level of precedence, left or right associativity is specified to define the order of evaluation of operators with the same precedence.

Error handling such as overflow, divide by zero check, and other exceptions in expression evaluation is not defined by the language.

Post-fix Expression	**Description**	**Associativity**
( )	Grouping Operator	left-to-right
++ -- !	Increment/Decrement by 1 limited to **for** stmt use expands to ID=ID+1. The operand is an l-value and the result is an r-value, with the side-effect of incrementing/decrementing ID by value of 1 Unary Negation is applicable to an arithmetic type only, the result is boolean TRUE if the value of its operand compares equal to 0, and FALSE otherwise	right-to-left
* / %	Multiplication, division, modulus are only applicable to arithmetic types. The '/' symbol yields the quotient, and the '%' operator the remainder of the division of the first operand by the second. If the second operand to division or modulus is 0 the operation is undefined.	left-to-right
+ -	Addition and subtraction are only applicable to arithmetic types. The '+' operator is the sum of the operands, while the '-' operator is the difference of the operands.	left-to-right
< <= > >=	Relational Operators are only applicable to arithmetic types all yield TRUE if the specified relation is mathematically correct, otherwise FALSE. The symbol '<' represents less, '<=' less than or equal, '>' greater than, '>=' greater than or equal'.	left-to-right
== !=	Equality Operators are only applicable to arithmetic types and produce a boolean type of TRUE if the mathematical relation is correct, otherwise FALSE. The '==' symbol is "equal to" while '!=' is "not equal to".	left-to-right
&&	Logical AND are only applicable to boolean types and also produce a boolean type of TRUE if both of the operands are TRUE, otherwise FALSE.	left-to-right
||	Logical OR are only applicable to boolean types and also produce a boolean type of TRUE if one or both of the operands is TRUE, otherwise FALSE.	left-to-right
=	Assignment requires a modifiable (not constant) l-value as left operand and it cannot be a function or an oscillator block. The type of an assignment expression is that of its left operand and the value is the value stored in the left operand after the assignment has taken place.	right-to-left

,	Comma operator is used to separate expressions, they are evaluated in order from left-to-right.	left-to-right

5. Definitions
Definitions specify the interpretation given to identifiers**.**

5.1 Primitive data types

Integers and floats are declared in the following manner:

int *identifier* = integer ;

float *identifier* = float ;

There are four different types of definition statements, each pertaining to a different element.

5.2 Oscillator Definition
There are two flavors of oscillator definition. One is for stand-alone oscillators, and the other is for oscillators associated with an oscillator bank.
osc *identifier = waveshape* ;
oscbank[*oscillator_index*] = *waveshape* ;

waveshape is one of the following: "SINE", "SQUARE", "SAW", "REVSAW". These are the four possible waves that an oscillator can output. The quotes are necessary. If an oscillator is not assigned a waveform in the declaration section, it is assigned a SINE by default.

5.3 Oscillator Bank definition
oscbank *identifier* = *value*

value is an integer representing the number of oscillators contained in the oscillator bank.

5.4 Envelope Definition
env *identifier* = {(*time*$_1$*, value*$_1$*)* , *(time*$_2$*, value*$_2$*), ... , (time*$_n$*, value*$_n$*)*} ;

(time$_k$*, value*$_k$*)* is a time-value pair. Each must be between 0.0 and 1.0. See the description of envelopes in the "Meaning of Identifiers" section. There must be at least one pair in each envelope definition.

5.5 Mixer Definition

mixer *identifier*;

Mixers are declared in the definition section, but actually assigned value in the Connection Section. In addition, mixers ignore segments. Any mixer assignment applies to all segments.

6. Statements

Statements are the smallest executable block of code in a programming language. They are logical sequences primarily executed to alter the state of a program by side effects.

6.1 Conditionals [Definition Section, Connection Section]

For loops and if/else statements may be used in both the definition and connect sections. In the definition section, the body of such loops may contain definitions or nested definition loops.

6.1.1 For loops:

```
for(assignment; bool_expression ; delta)
   {
      definitions  | nested conditionals
   }
```

assignment - must be of the form: *identifier = (int | float)*
   The identifier represents a variable not yet defined in the program. It's scope lasts throughout the body of the following loop.
bool_expression - an expression that evaluates to true or false. See the section on "Expressions".
delta - must be in one of the following three forms: *identifier = expression*
identifier++   *identifier--*
   The for loop in AASL acts as one would imagine it to. The assignment initiates a condition. Delta is performed at the end of each loop. Once delta has been performed, the condition_expression is checked and, if it is false, the loop is exited.

6.1.2 If/Else statements:

```
if(bool_expression)
   {
     definitions  | nested conditionals
   }
[else
   {
     definitions  | nested conditionals
   }]opt
```

NOTE: The braces denote an optional section and are not ever present in an actual if/else block.

6.2 Connection Statements [Connection Section]

There are two kinds of connection statements. One kind involves oscillators. The other involves mixers.

6.2.1 Oscillator Connections:

oscillator(freqValue, ampValue) ;	assigns constant values to frequency and amplitude
oscillator(freqValue, {oscillator | envelope | function_call}) ;	assigns a constant value to frequency and assigns an element/function to control amplitude
oscillator({oscillator | envelope | function_call}(central_frequency), ampValue) ;	assigns an element/function to control frequency and a constant value to amplitude
oscillator({oscillator | envelope | function_call}(central_frequency), {oscillator | envelope | function_call}) ;	assigns elements/functions to control both frequency and amplitude.

NOTE: In the above descriptions, curly braces are used for grouping and would not appear in an actual connection statement. All parentheses shown, however, would appear.

- *oscillator* - is either of the form *identifier* representing a stand-along oscillator or of the form *identifier*[*integer* | *identifier*] (<--describe 'identifier' type in oscbank) representing an oscillator element from a given oscillator bank.
- *freqValue* - is an integer or a float equal to the assigned frequency in Hz. A mathematical expression which evaluates to an int or a float, or oscillators and envelopes, may be inserted here.
- *ampValue* - is a float between 0.0 and 1.0 equal to the assigned amplitude relative to maximum volume of 1.0. A mathematical expression which evaluates to an int or a float, or oscillators and envelopes, may be inserted here.
- *central_frequency* - is an integer or float equal to the central frequency around which the attached element modulates the frequency. The maximum range of variation is a single octave, which corresponds to a minimum of half the central_frequency and a maximum of double the central frequency. As an example, if an oscillator o1 putting out a 1Hz SINE wave at full amplitude is attached to the frequency input of an oscillator o2 and a central frequency of 440 Hz is assigned, then the output of o2 will shift smoothly between 220 Hz and 880 Hz. The instantaneous frequency of the output of o2 will be 440Hz at every zero crossing of o1's sine wave.
- *function_call* - as explained in the "Functions" section, all functions may be thought of as returning complex oscillators **(??)**. That is, a function call may be used in the "connect" section wherever an oscillator could appear. When used as an argument, the function would control the corresponding frequency/amplitude.

6.2.2 Mixer Connections:

mixer = term + term + ... + term ;

- *mixer* - an identifier representing a mixer defined in the "def" section
- *term* - of the form *(factor * output*) where *factor* is an integer or float and *output* is an oscillator or a function call.

Each term represents an input. The inputs are mixed in ratios proportional to their factors.

6.2.3 Connection conditionals

These are identical to the "def" section conditionals described in section 3.2.1.2 with the exception that the bodies must be composed of connection statements and connection conditionals.

6.3 Output Equation:

OUTPUT = *mixer_connection* ;

mixer_connection - identical to the mixer connections described in 3.2.2.1.

7. Functions

Functions are smaller, self-contained portions of programs that perform specific tasks. When using real synthesis modules, elements are first designed then connected to one another in various ways. AASL functions reflect this practice by having a definition section and a connections section, followed by an OUTPUT assignment statement. They are used in place of oscillators - in fact, they boil down to multiple oscillators treated as a whole. Again, within each section, **different kinds of statements are allowed.** See Section 6, Statements, for more information.

```
start func identifier
      definition section
      connection section
      output statement
end func identifier
```

7.1 The Definition section

The definition section is where the synthesizer's elements are declared.

```
start def
  (definitions | definition conditionals)*
end def
```

7.2 The Connection section

The second section is the "connect" section, which contains details about the element's characteristics and how elements are interconnected.

start connect

    *(connection statement | connection conditional)**
end connect

7.3 Return Value
The return value from any user defined function is a wave form, which can replace an oscillator. The output is defined as:

OUTPUT = *expression* ;

The expression must evaluate to a valid oscillator.

8. Scope

8.1 Blocks:

A *block* is a section of code surrounded by braces { }. Block defines the scope, visibility, of any declared variable.

8.2 Scope:

The *scope* of an identifier is the range of the program in which the declared identifier has meaning. Scope is determined by the location of the identifier's declaration.

8.2.1 Block Scope:

An identifier appearing within a block has *block scope* and is visible within the block, unless hidden by an inner block declaration. Block scope begins at the opening brace ({) and ends at the closing brace (}) completing the block. If there is an identifier with the same variable name in a block above the defined block. Then the identifier with in the innermost block takes precedence.

8.2.2 Function Scope:

An identifier declared within a function has function scope and it should be unique throughout the function in which it is declared.

9. Anatomy of an AASL Program
Every AASL program consists of a header section and a main function followed by optional user-defined functions.

9.1 Header Section
An AASL program begins with a short header section. Every header begins with "start header" and ends with "end header." Within the section, one assigns the name of the output file, the length in seconds of the tone to be generated, and optional assignment of the number of segments the tone is to broken up into. This segmentation is used to alter

the tone as time progresses through the use of conditionals.

-Output Assignment
   OUTPUT =" *filename*";
filename is the name of the file to which the generated tone will be written.

-Tonelength Assignment
   TONELENTGH = *value*;
value is an integer of float that gives the desired length of the tone in seconds

-Segment Assignment (optional)
   SEGMENTS = *value*;
value is an integer representing the number of equal-length time slices into which the tone will be split. Associated with the number of segments is a keyword SEG which evaluates to an integer representing whichever segment the tone is currently in. The idea of segments is used in writing conditional statements. This is best understood through an example.
Assume our header contains the following lines:

```
TONELENGTH = 10;
SEGMENTS = 2;
```

We may now view our tone as being divided into 2 five-second sections. If we want an oscillator, o1, to vary between SINE and SQUARE wave output, we could write the following in our "def" section:

```
if(SEG  == 1)
    {o1 = SINE;}
else
   {o1 = SQUARE;}
```

9.2 Main Function

All AASL programs must have a main function, which simply follows the form of all functions, except that it has the identifier "main". It must follow the Header section.

9.3 User Defined Functions

User defined functions can have any valid identifier, and they must follow the main function.

10. Sample Program

```
/* Header Section */
start header
OUTPUT = "testsound"
TONELENGTH = 10 //10 second sound
SEGMENTS = 5
end header

/* Main Function */
start func main
```

```
       /* Definition section */
         start def
            mixer S1;
            mixer S2;
            osc o1 = SINE; //defines a sine wave oscillator, o1
            osc o2 = SQUARE;
            oscbank ob1 = 3; //bank of three oscillators

//set waveform of oscbank elements depending on which
//time segment you're in
            if(SEG == 0)
            {
               ob1[1] = "SQUARE";
               ob1[2] = "SINE";
            }
            else
            {
               ob1[1] = "SINE";
               ob1[2] = "SAW";
            }

//third osc in the bank does not depend on SEGMENT
            ob1[3] = "SINE";

            env e1 = {(0.0,0.3)(0.2,0.2)(0.4,0.2)(0.8,0.8)(1.0,0)};
            //defines envelope w/ (time, amp)
            env e2 = { (0.0,0.0) (0.3,0.0) (1.0, 1.0) };

         end def //end definitions

      /* Connection Section */
      start connect

         o1(440, e1);
//second oscillator in oscbank controls amplitude
         o2(e2@1000, ob1[2]);
//for loop featuring .size feature (returns # of osc in oscbank
        for(x=1; x<; x++){
            if(x==1 || SEG==2 || SEG==5)
            {
               ob1[x](440, 0.5); //apostrophes differentiate variables
            }
            else
            {
               ob1[x](220*x, ob1[x-1]);
            }
        }

         S1 = 0.4*o1 + 0.6*o2;
         S2 = 2*S1 + ob1[1] + 0.5*ob1[2] + 0.25ob1[3] + 0.25*o2o{1000};
         //note one of the ouput signals is function call

     end connect

      /* Output Section */
      OUTPUT = S1 + o5;
```

```
   end func main


  /* User defined funtion...takes one integer as argument */
  start func o2o (int x)

    start def
        osc o1(SAW);
        osc o2(SINE);
     end def

     start connect
        o1(x, o2);
        o2(0.5*x, .75);
     end connect

     OUTPUT = o1;

 end func o2o

  /* End of Sample Program */
```

Chapter 4

Project Plan

Engineering Process

We had regular group meetings to discuss the project progress, divide the next set of tasks amongst the team members, and set goals for the preceding week. Initial weeks saw more frequent and longer meeting as the scope and the exact requirements were only understood by Rob (Team Lead) and the concepts and exact requirements of the tone generation had to explained to other members. Later during the middle of the semester the meetings became shorter and somewhat more infrequent during the mid-terms. At times, it was not possible for a member to attend a meeting. To keep every member updated, all the changes were communicated through the common group mailing list, comprising of all the team members. For one on one communication, instant messaging tools like AOL messenger and gtalk were used for conversation.

Telling every team member what to do is unlikely to lead to the most efficient allocation of human resources. Instead, we allowed the fear of failure - the incentive of the PLT "market" - to guide each team member to where their motivation and talents fit best. This is the philosophy we followed. Whoever was skilled in a particular section of the compiler the most, did the most in developing the Aasl compiler. This is where he fitted at the time. Over time, roles shifted as availability and energy fluctuated among team members due to their class schedules, but our fluidity allowed the invisible hand to create the most efficient team possible at the time.

Programming style guide

Code Conventions for the Java Programming Language (revised April 20, 1999)
http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

Project timeline

	Docs	Lexer (AaslGram.g)	Parser (AaslGram.g)	Tree Walker (walker.g)	Intermediate Types	Code Generation	Tests
15-Jan							
22-Jan							
29-Jan							
5-Feb							
12-Feb	Proposal (2/7)						
19-Feb							
26-Feb							
5-Mar	LRM						
12-Mar							

19-Mar							
26-Mar		Created. Basic Lexer rules (1.4)		Created			
2-Apr		Function calls (1.8)			Created Package		Created (JUnit tests)
9-Apr		1.23	Created.	"or", "and" (1.7)		Created (1.1)	1.6
16-Apr			1.43	SEG (1.9)		Minor changes (1.5)	1.18
23-Apr			1.5	Dynamic scoping (1.46)		Comments (1.8)	
30-Apr			1.59	Fixes (1.54)		Included functions (1.10)	Many tests added (1.21)
7-May	Final Report		1.89	1.110	1.13	1.27	Ad hoc

Team Roles and Responsibilities

Name	Responsibilities
Robert Katz	Visionary, team leader, back-end
Carlos Rene Perez	Second in command, Front-end: grammar, tree-walker, intermediary types, user-func.
Vaishnav Janardhan	Front-end: Control flow, tree-walking, data types and programming support
Albert Tsai	Front-end: grammar, Testing and documentation

Programming Guide and Software Development Environment

All the phases were carried out by the respective team-member on his laptop or CLIC lab machine. Since both Antlr and Java are platform independent, each member was free to use Windows, Linux or Mac OS. No problems were encountered due to programming in multiple platforms. Also, almost every team member worked from home, or in the lab of his choice. The version control system, installed in the Clic lab machines was used as CVS repository. Logins were created and access was given to all members of the team to update the code or the document independently. The CVS plugin for Eclipse was used and it got integrated into the Eclipse IDE, which helped in easier modifications to the

code. During the final phases Junit 3.8 was used for doing unit and system testing.
The back end code was written entirely in Java, while the front end code is written in ANTLR. In general, there were often overlap of participation between the front-end parsing and back-end coding. Not only did every team member work on the specific assigned piece of the source code in a particular phase of the project but also each one participated in several works simultaneously to ensure the whole project is agreed upon. For example, as the team leader, Rob had to develop the skeleton code for both the front-end and the back-end to make sure that there are no inconsistencies as it developed and progressed.

Operating System	Windows XP, Linux, Mac OS
Java Runtime Environment	Java 1.5.0 with Generics
Development platform	Eclipse 3.2
Antlr	ANTLR 2.7.6 (via Eclipse plugin)
Version control system	CVS repository, CVS -1.11.17
Unit testing	JUnit 3.8
Antlr tutorial	Jeremy D. Frens's ANTLR Unit testing framework [http://cs.calvin.edu/curriculum/cs/382/jdfrenscompilers/parser.html]

Java
With the exception of the Lexer, Parser, and Walker, which were written in ANTLR, all of the other components of the project were written in Java. We utilized Java object oriented philosophy to create well-defined classes and methods. The version of the Jave Runtime Environment used for this project is Java 1.5.0. We also followed the java programming convention at http://java.sun.com/docs/codeconv/, and constantly referred Java 2D API documentation at http://java.sun.com/j2se/1.5.0/docs/api/

Operating system
As Java, which is is a platform independent language was used for the language development, all the team members used the operating system of their choice like Windows Xp, Linux and Mac OS.

ANTLR (ANother Tool for Language Recognition)

ANTLR is a language tool that provides a framework for constructing recognizers, compilers, and translators from grammatical descriptions. We used ANLTR to implement our Lexer and Parser, and used ANLTR-generated codes to construct the AST (abstract syntax tree) and to create a tree walker to traverse through it.

IDE

Eclipse SDK 3.2 was used to facilitate the development process. The ANLTR plug-in for Eclipse that we used is ANTLR 2.7.6.

CVS

CVS is a revision control system that was used to organize our project. The CVS Repository was hosted on a CRF Linux machine and was accessed both remotely and from the CLIC lab.

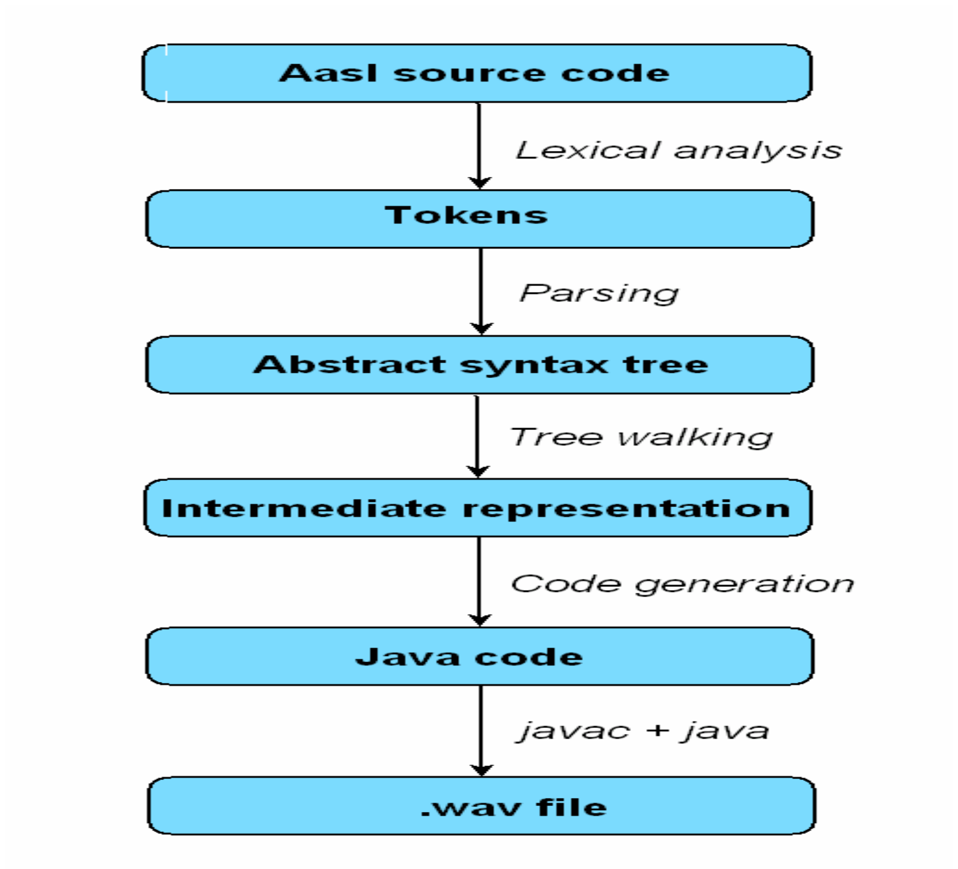
Chapter 5

Architectural Design

In this section we describe the structure of the aasl source program, the design and structure of the Aasl compiler.

Architecture of Aasl compiler

Aasl has mainly five inter-related components: Lexer, Parser, Walker, Code generation and the output java source file. The first three are in the file Aaslgram.g and walker.g file. The error handling code is embedded in the tree walker file. The flow of information between components is as shown in the block-diagram below.



- The lexer reads the Aasl program source file and translates the stream of characters into a stream of tokens and then output tokens to the parser.
- The parser analyzes the syntactic structure of the program and translates the stream of tokens into an Abstract syntax tree (AST).

- Walker reads the different tree structures constructed in the AST and does a semantic analysis of the program, by checking for errors in the program and check if the program elements and their connection are correct.
- Code generation module will take the output from the tree walker and generate the java source file. The generated java file will be compiled and executed to create the required .wav file.

Lexical analysis:

The Lexer (AaslGramm.g) is implemented in ANTLR. In the lexer all the different source program token are identified and the unnecessary objects like the empty space are removed and the plain stream of tokens is given as input to the parser. Lexer is composed of expression operator, identifier definitions, key words, including comments, semicolon and others.

1. **Expression operators:** The tokens defined to be identified for operation on expressions.

```
PLUS : '+' ;          MINUS : '-' ;     TIMES : '*' ;
DIV : '/' ;           MOD : '%' ;       ASSIGN : '=' ;
NOT : '!' ;           LT : '<' ;        GT : '>' ;
LE : "<=" ;           GE : ">=" ;       EQ : "==" ;
NE : "!=";            AND : "&&" ;      OR : "||" ;
INC : "++" ;          DEC : "--" ;      COMMA   : ',' ;
SEMIC   : ';' ;       DQUOTE  : '"' ;   LPAREN  : '(' ;
RPAREN  : ')' ;       LBRACE  : '{' ;   RBRACE  : '}'
LBRACK : '[';         RBRACK : ']';     AT : '@';
```

2. **Identifier definitions:** In Aasl the identifiers are consists of letters, digits and underscore and the first letter has to be an alphabet.

```
ID options {testLiterals = true; }
  : (LETTER | '_') (LETTER | DIGIT | '_')* ;
```

3. **Key words:** Aasl has many keywords to identify the mixers, oscillator, oscillator banks and the envelope.

```
tokens
{
      OSC = "osc"
      OSCBANK ="oscbank"
      ENVDEF ="env"
      MIXERDEF ="mixer"
      …………
}
```

4. **Comments:** C-style single line and multi line comment is used.

```
SL_COMMENT
      :    "//" (~('\n'|'\r'))* {$setType(Token.SKIP);}
      ;
```

```
COMMENTS      :    "/*"
      (
            { LA(2)!='/' }? '*'
            |    '\r' '\n'       {newline();}
            |    '\r'           {newline();}
            |    '\n'           {newline();}
            |    ~('*'|'\n'|'\r')
   )*
   "*/"        {$setType(Token.SKIP);}
;
```

Parsing:

Aasl Parser uses the output of the Aasl lexer to create an abstract syntax tree that represents the grammatical constructs in the source program. Any syntax error will be detected during the parsing phase while constructing the abstract syntax tree. If any expression in the source program does not match the tree constructs, then an error will be thrown to the user and the program will exit.

During the parsing phase, the AST is constructed for the conditional branches (i.e. "if" and "for"), Oscillators, Oscillator banks, Mixers and envelopes. Separate AST are constructed for the definition and connection section.

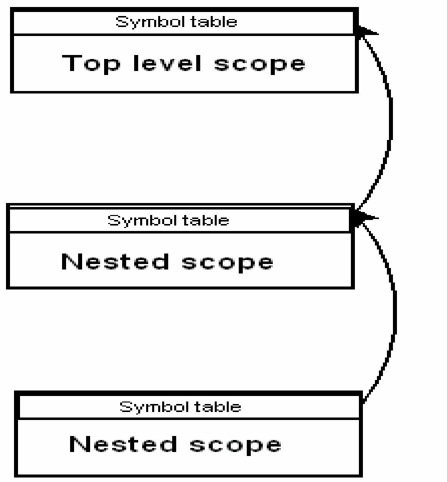
Tree walking / Static semantic analysis:

The AST represents all the AASL elements like the mixer, osc, oscbanks and others as a branch with root as the keyword and children as arguments. When walking the AST, if there is a root element that matches any keyword, the walker will call the related functions from the back-end in classes defined and pass the required child nodes as corresponding arguments to the functions.
In the back-end the classes are defined for all the Aasl data types and the different objects of the Aasl language. During tree walking all the different expressions would be resolved and the AST for each of these would be constructed and walked through. While walking each one of these trees, different arguments would be resolved and they would be passed over to the class of its individual type to do a static and semantic analysis. For each and every data type and individual classes have been created. All the data types extend the super class AaslType, so that the data type could be passed around as AaslType and resolved during the final computation or error checking. In the AaslType the arithmetic operations like "+", "-", "*", "/", ">", "<" and others are overloaded to operate on specific data types. During parsing separate AST are constructed for the elements in definition section and the connection section. One unique feature in Aasl is that for oscillator bank data type is formed of array of oscillators. When oscillator bank is defined, there is no actual oscillator bank object stored but the individual oscillator elements are stored in the symbol table and the vector. All the oscillator bank elements would be de-marked with the "$" de-limiter to differentiate them from other oscillator elements. The same "$" de-limiter would be used to search for the element also. If any element used in the definition section is not previously defined, then that object will be defined and added to the symbol table and the vector table. In the tree walking some of the unique semantics of Aasl is verified, like the wave type is checked to be either one of these type,

"SINE|SQUARE|SAW|REVSAW". The amplitude input to any oscillator is verified to be within the range of 0 to 1 for oscillator definitions, as the amplitude input above 1 would make the output tone inaudible. During semantic analysis when another oscillator is used to control the frequency input of the base oscillator, we have added a check to make sure the user gives a central frequency around which the input oscillator varies the frequency so that base oscillator frequency doesn't go below the audible frequency.

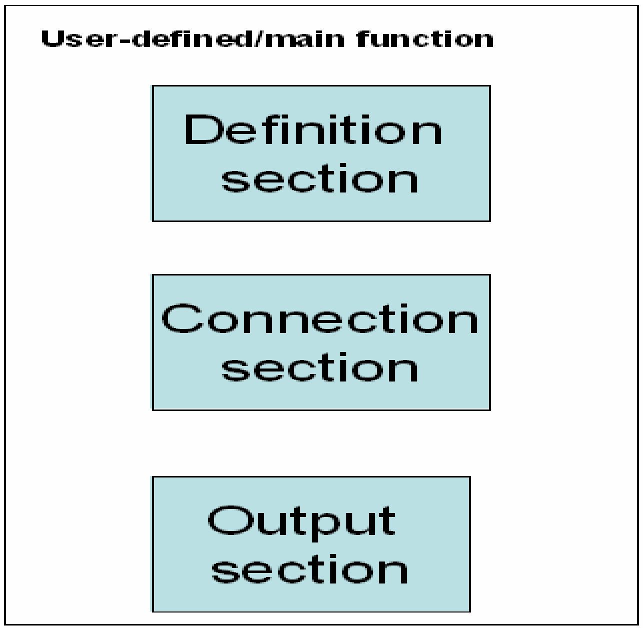
Control Flow
The branch instructions are supported in Aasl are "if" and "for". In our Aasl "if" is very unique when compared to the general programming languages. "if" conditional branch is used to set different input variable parameters for different segments. For all the segments set in the "if" conditional branch statement, if body statements inside the then branch would be executed. And for all the segments that are not set in the "if" statement, the segment array would be flipped and the else branch instructions would be executed for the rest of the segments. For the "for" statement, the tree structure will have the assignment, bool, delta and the for-body segments as children of the "for" body root. All these elements would be passed over to the FoLoopInstruction.java class and will be executed in run function and the final output would be returned back.

Symbol table:
The symbol table keeps track of the name, the type and the scope of all the identifiers used in the Aasl programs. Symbol table is a simple data structure abstracted using a Hash table, which stores the name, value pair. This simple symbol table is very useful in adding new elements and also in looking up of stored elements.



For providing nested scoping rules, the symbol table is constructed to form a linked list of symbol tables with each symbol table pointing to its next level of symbol table for dynamic scoping rules. When ever the program enters a new scope, a new symbol table is created and added to the linked list of symbol tables.

User-defined functions:



Users are given the option to define their own functions, to bringing in more modularity to the user written programs. All the functions should be defined before the main function and the user-defined functions can only take integer or a floating point number as arguments. But they need to have one or more arguments, except for the main function which dosen't take in any input parameters.

The structure of any Aasl defined function would be as shown in the figure, with a definition section, connection section and the output section. All the Aasl elements, variables would be declared in the definition section and variables used in the rest of the program will all have to be declared in the definition section. The elements declared in the definition section, would be connected to multiple elements in the connection section. The elements connected in connection section would output the resultant waveform as mixer. The mixer output in the main function would generate the .wav file as the final output.

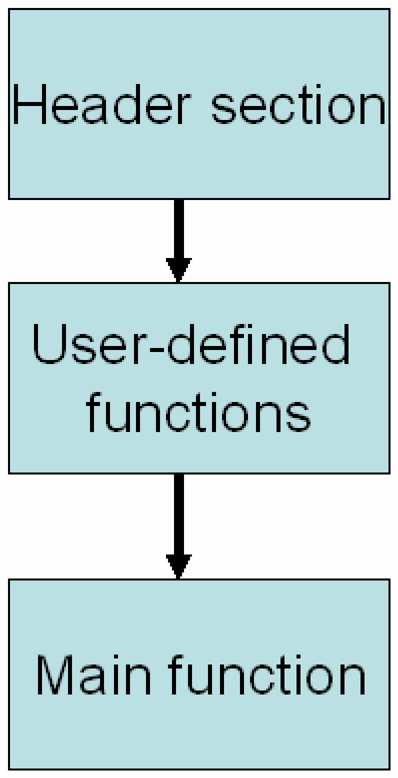
The tree-walker while walking the AST will keep loading the identifiers into the vector table. The first few vector elements would have the output file name, tone length, no of segments and the rest of the vector table will have all the identifiers declared in the program.

Code-generation:

Code generation is the core functionality of the Aasl compiler, where the actual java file would be generated for the components used to generate the tone in the Aasl source program. The vector table received from the tree walking is used in code generation. All the identifiers would be checked to be of Aasl data type and when it hits a mixer it will recursively check to make sure that all the elements in the vector are of Aasl data type. After the vector check, the functions would be removed from the vector table. The removed Aasl elements would be checked for any looping cycles among the interconnected elements. Cyclic check makes sure that no oscillator output is indirectly connected to one of its input. The java output file generated will have the header section, user-defined function, the envelopes declared followed by rest of the mixers and oscillators. Among the oscillators and mixers the oscillators with constant frequency and amplitude would be declared first to do a semantic check of the connected elements. Only the mixers and oscillators would vary across segments but the envelopes would be constant across segments. In the end, the tail section would be defined for creation of the final output .wav file.

Structure of the source program

Aasl source program is divided into three main sections: Header section, User-defined functions and the main function. Each and every section is clearly marked with the start and the end tags for each section. All programs written in Aasl should follow the same programming construct, with the header section in the beginning, followed by user-defined functions and the main-function. All the user-defined functions must be defined before the main-function.



Aasl program structure

Header-section:

Header section defines the name of the output file, the tone length and the total number of segments in the out put wave file. Output file declared would be the output file generated with .wav extension, which can be played using some third party application. The tone length is defined in seconds, which gives the total duration for tone outputted into the wave file. No of segments declared would be the total number of segments the out put tone would be divided into.

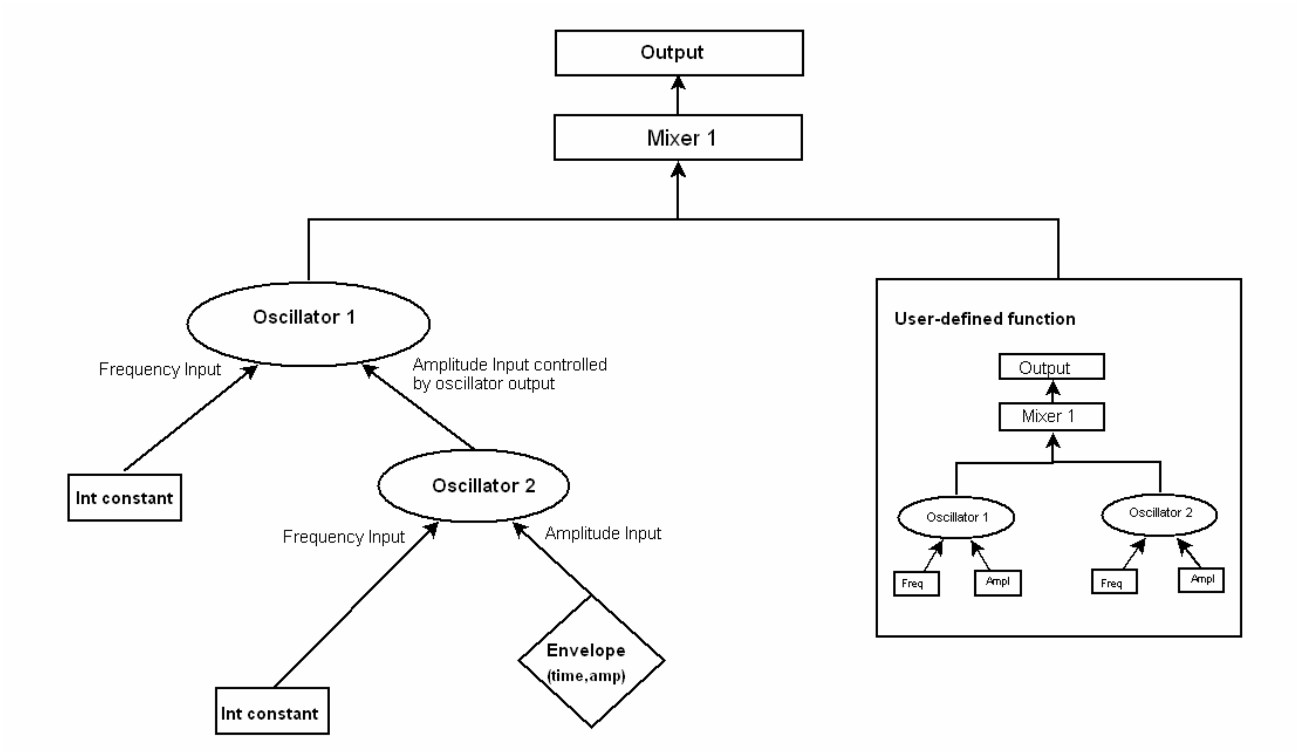
User-defined functions:

User can define the functions of their own as part of the Aasl source program. All the user defined functions would have to be defined before the main function. The defined user functions could be used as input to the mixer in the connection section.

Main function:

Main function would be start with the identifier "main", identifying the main function. Each user program can have at most one main function and the program execution would start from the main function.

Control flow of a typical Aasl program would be as shown in the figure.



Sample user program

As seen in this figure, Aasl compiler follows bottoms up approach. All the basic components should be declared and connected before connecting the higher level components, ex. Envelope with time and amplitude input should be defined and connected to the oscillator before connecting it to the oscillator2. Similarly, oscillator 2 input should be connected before connecting oscillator 2 as an input varying the amplitude of the oscillator 1. This representation of the Aasl program clearly helps the user in identifying the loops early on, before the compiler can throw an error during the compilation phase.
The Aasl compiler would compile the Aasl source file, also execute and finally generate a .wav file as out put with the tone that was defined in the aasl source file. The generated .wav file should be played back using some third party application.

Chapter 6

Test Plan

Automation

String Comparison

ANTLR is able to output the parse trees it generates with a given parser/lexer. These parse trees are in a simple text format, and they mention every production rule used to arrive at that tree. Knowing this, a test harness was created aimed at making new tests as simple as adding files to the directory (no recompilation of code would be necessary). The test harness would scan a directory containing the tests. Each test would be composed of two files, an arbitrary file named *n*, and its corresponding file *n.out*. *n* contains code that we want to test the parser/lexer with. *n.out* contains the formatted parse tree that we expect the parser to return. Therefore it is a simple matter to add a new test by creating a new <*n,n.out*> pair of files and dropping them into the test directory.

Unfortunately, with even a moderate amount of rules, the created parse tree is enormous. It became too slow to create full parse trees (which included EVERY rule used in production) for substantial test code. A better test framework would merely use parse trees composed of tokens, and the trees explicitly returned by some rules (i.e., rule: yadda yadda {#tree = #([RULE, "RULE"], rule}). Enter Jeremy D. Frens (JDF)'s (of Calvin College) test suite...

JDF's JUnit+ANTLR Framework

[http://cs.calvin.edu/curriculum/cs/382/jdfrenscompilers/unittestantlr.html]
ANTLR's unit testing capabilities are pretty poor. In fact, Terence Parr identified unit testing capabilities as one of the things he would like to add for ANTLR 3.0 (see his gUnit proposal on http://www.antlr.org/summer-of-code/2007/index.html). Being able to leverage the JUnit framework for the compiler would be a tremendous boon - unit testing is part of good software engineering. Luckily, Jeremy Frens at Calvin College has created

a test harness which utilizes JUnit 3.8 to conduct lexer/parser/tree walker unit tests. Tests follow the JUnit 3.8 template, where each test is a function. Each function takes in a string input, runs the lexer/parser/tree walker against it, and compares the generated parse tree against the expected output. Although this sounds similar to the String Comparison harness, JDF's harness reduced the ANTLR parse tree greatly (while preserving correctness), before doing a comparison with the expected tree. This made tests a lot easier to write. Eventually we wrote about 50 test cases using this harness.

From here on, testing took a more ad hoc, integration style form. If step *n* worked, we assumed steps 1 to *n*-1 worked.

Code Generation Tests

Before the tree walker was even started, we started code generation testing by manually creating sample intermediate representations, and passing them to the code generator. When the code generator produced the correct output (a compilable, executable .java file utilizing Java's sound libraries), we were satisfied.

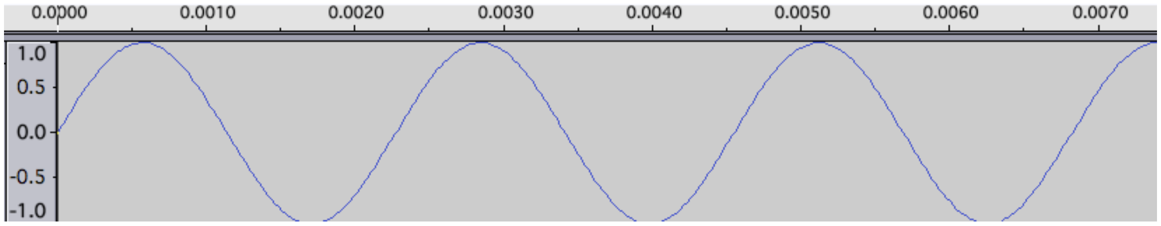
Sample AASL Code tests

Yet another test harness was created which scans for input files in a directory, assembles a list of valid files to compile, and then attempts to compile each one. What follows are some sample tests, and the corresponding .wav that we expected them to produce:

Test 1:

```
-------------------------------------------
start header
OUTPUT = "helloworld"
TONELENGTH = 3
end header

start func main
      start def
            oscbank oscb1 = 2;
            oscb1[0]="SINE";
            oscb1[1]="SQUARE";

            mixer s1;
      end def

      start connect
            oscb1[0](0.5,1.0);
            oscb1[1](oscb1[0]@440,1.0);

            s1 = oscb1[1];
      end connect

      OUTPUT = s1;
end func main
-------------------------------------------
```

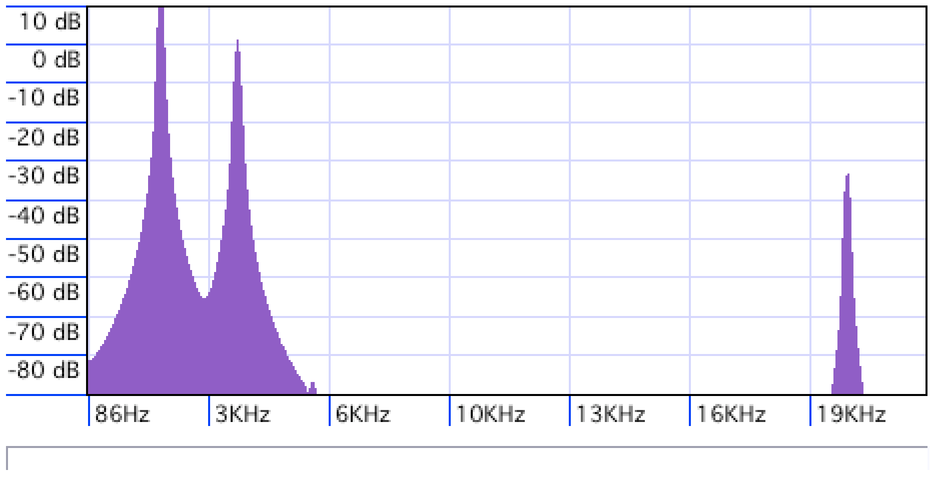
Output:



Test 2:

```
-------------------------------------------
start header
OUTPUT = "factortest1"
TONELENGTH = 3
end header

start func o3(int x)
    start def
        osc o3="SINE";
    end def

    start connect
        o3(x,1.0);
    end connect

    OUTPUT = o3;
end func o3

start func main
    start def
        int x=2000;
        int y=3*x;
        osc o1 = "SINE";
        osc o2 = "SINE";

    end def
    /*
    * ha ha comments
    */
    start connect
        o1(x,1.0);
        o2(2*x,1.0);
    end connect

    OUTPUT = 200*o1 + 50*o2 + o3{20000};
end func main
-------------------------------------------
```

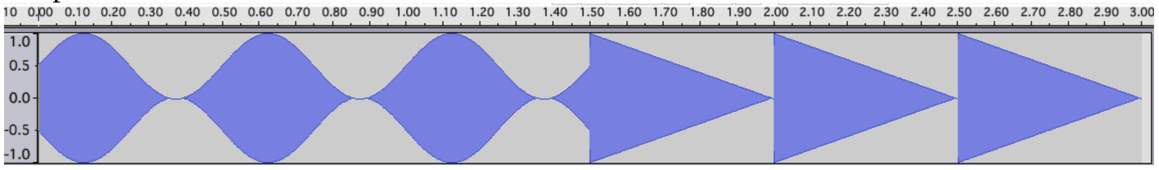
Output:



Test 3:

```
--------------------------------------------
start header
OUTPUT = "segtest1"
TONELENGTH = 3
SEGMENTS = 2
end header

start func main
    start def
        //osc o1 = "SINE";
        //osc o2 = "SQUARE";
        oscbank oscb1 = 2;
        if(SEG==0){
            oscb1[0]="SINE";
        }
        else{
            oscb1[0]="SAW";
        }

        oscb1[1]="SQUARE";

        mixer s1;
    end def

    start connect
        oscb1[0](2,1.0);
        oscb1[1](oscb1[0]@440,oscb1[0]);

        s1 = oscb1[1];
    end connect
    OUTPUT = s1;
end func main
-------------------------------------------
```

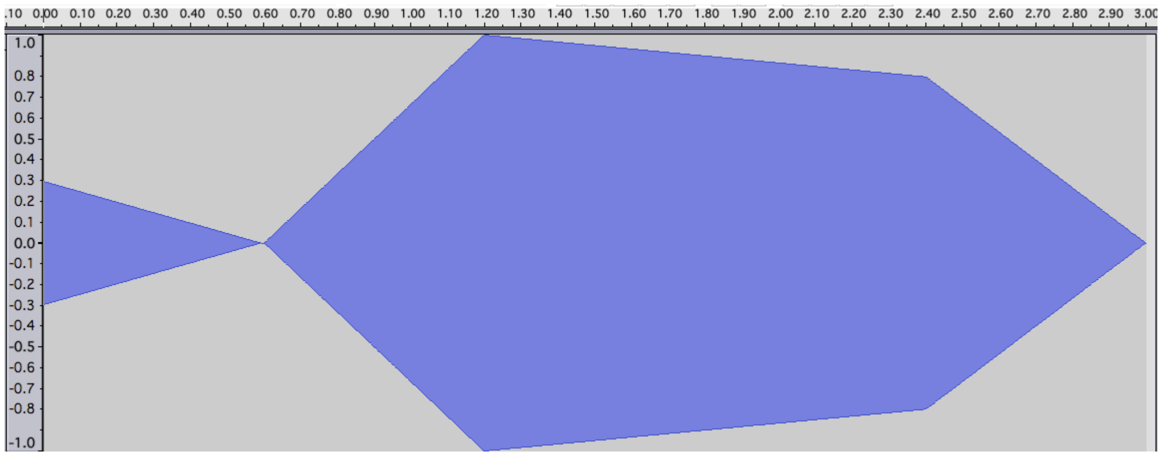
Output:



Test 4:

```
-------------------------------------------
start header
    OUTPUT = "envtest1"
    TONELENGTH = 3
    SEGMENTS = 2
end header

start func main
    start def
        oscbank oscb1 = 2;
        if(SEG==1){
            oscb1[0]="SINE";
        }
        else{
            oscb1[0]="SAW";
        }

        oscb1[1]="SQUARE";

        env e1 = { (0.0,0.3) (0.2,0.0) (0.4,1.0) (0.8,0.8) (1.0,0.0) };

        mixer s1;
    end def

    start connect
        oscb1[0](2,1.0);
        oscb1[1](oscb1[0]@440,e1);

        s1 = oscb1[1];
    end connect

    OUTPUT = s1;
end func main
-------------------------------------------
```

```
Output
```



Chapter 7

Lessons Learned

Robert Katz
By far the most important advice I have is to spend a great deal of time with older projects. Even if they are cryptic and illegible at first, put effort into following exactly what is going on. I consider it to have been invaluable that, within the first month, I had spent a couple hours each with a few different past projects. I have a few words for those who would be team leaders. First of all, delegate wisely, but make sure you assign the basic, important things to yourself. For example, if your LRM is due on Monday, make sure you are the one ensuring it actually gets submitted. Also remember that things get frustrating sometimes and, though it may be excusable for people to lose their cool every so often, a team leader really can't do this.
Also, don't give Rene your laptop. You may never see it again.

Carlos Rene Perez - Having worked on different software engineering paradigms (iterative, avalanche) we decided to implement a concurrent interative design philosophy. Looking at our results and with my previous experience I can say that it doesn't matter which design philosophy you use.

Vaishnav Janardhan - Assign clearly delineated responsibilities to people and hold them accountable for it.

Albert Tsai - Consistent periodic contributions are a lot more valuable than sporadic, extended bursts of effort.

Chapter 8

Appendix

Source code

AaslGram.g

```
/*
 * AaslGram.g
 * Written by Rob Katz, Carlos R. Perez, Vaishnav Janardhan, Albert Tsai
 */

header
{
	package aasl;
}

/*
 * ===================================================================
 *                                                       PARSER
 * NOTES:
 * For Albert's tests, use:
 * class AASLParser extends Parser("antlr.debug.ParseTreeDebugParser");
 * comment out:
 * classHeaderSuffix=org.norecess.antlr.TestableParser;
 *
 * For Carlos's tests, use:
 * class AASLParser extends Parser;
 * classHeaderSuffix=org.norecess.antlr.TestableParser;
 *
 * ===================================================================
 */

//class AASLParser extends Parser("antlr.debug.ParseTreeDebugParser");
class AASLParser extends Parser;
options{
	k=3;
	buildAST = true;

	classHeaderSuffix=org.norecess.antlr.TestableParser; /* needed for jUnit
test */
}
tokens{
	PROGRAM;
	HEAD;

	DEFSTATEMENTS;
```

```
      INTDEF;
      FLOATDEF;
      MIXERDEF;
      OSCDEF;
      ENVDEF;
      OSCBANKDEF;
      OSCBANKELEMENT_DEF;

      EMPTYMIXERDEF;
      EMPTYOSCDEF;
      EMPTYENVDEF;
      EMPTYINTDEF;
      EMPTYFLOATDEF;

      CONSTATEMENTS;
      OSCCON;
      OSCBCON;
      FUNCCALL;
      FUNCDEF;
      EXPR_LIST;
      ARG_LIST_DECL;
      ARG_LIST;
      NEGATE;
      SEGSTMT;
      IFCON;
      MIXSTMT;
      BANKINDEX;
      OSCCONARG1;
      OSCCONARG2;
      OSCBANK;


      /* Literals Hash Table */
      START = "start";
      END = "end";
      FUNC = "func";
      MAIN = "main";
      DEF = "def";
      CONNECT = "connect";
      OUTPUT = "OUTPUT";
      TONELENGTH = "TONELENGTH";
      SEGMENTS = "SEGMENTS";
      SEG = "SEG";
      IF = "if";
      FOR = "for";
      SINE = "SINE";
      TRUE = "true";
      FALSE = "false";
}
/* extra parser code required for jUnit test */
{
  public int nr_error = 0;
  private boolean myFailure = false;
  private boolean myQuietErrors = false;

  public void reportError( String s ) {
        super.reportError( s );
        nr_error++;
    }
  public void reportError(RecognitionException ex) {
    if (shouldPrintErrors())
      super.reportError(ex);
    myFailure = true;
    nr_error++;
  }
  public boolean successfulParse() { return !myFailure; }
  public void atEOF() {
    try {
      match(EOF);
    } catch (Exception e) {
      myFailure = true;
    }
  }
  public void setQuietErrors(boolean quietErrors) {
    myQuietErrors = quietErrors;
  }
  private boolean shouldPrintErrors() {
    return !myQuietErrors;
  }
}

/******************************************
 * 4 EXPRESSIONS
 ******************************************/
// testAssignment()
assignment: ID ASSIGN^ (expr|STRING);

// testBoolExpr()
bool : join (OR^ join)* ;
join : equality (AND^ equality)* ;
equality : ( rel ((EQ^ | NE^) rel)* )
                    | (SEG^ EQ! expr);
rel      : expr ((LT^ | LE^ | GT^ | GE^) expr)* ;
expr     : term ((PLUS^ | MINUS^) term)* ;
term     : unary ((TIMES^ | DIV^) unary)* ;
unary    : MINUS^ unary { #unary.setType(NEGATE); } | NOT^ unary | factor ;
factor   : LPAREN! bool RPAREN!  | INT^ | FLOAT^ | ID^ | TRUE^ | FALSE^;


/******************************************
 * 5 DEFINITIONS
 ******************************************/
//[DEFINITION SECTION]
// testDefs()
defs : ( mixdef
|   oscdef
|   envdef
|   oscbdef
|   oscbelementdef
|      assignment
|      intdef
|      floatdef
  ) SEMIC!;

defstatements :  (defs | conditionaldef)+
       {#defstatements = #([DEFSTATEMENTS,"DEFSTATEMENTS"], defstatements);}
                    ;

/* ****************************************
 * 5.1 PRIMITIVE DATA TYPES
 * ****************************************/
intdef : "int"! ID (ASSIGN! expr {#intdef = #([INTDEF,"INTDEF"], intdef);} |
(COMMA! ID)* {#intdef = #([EMPTYINTDEF,"EMPTYINTDEF"], intdef);});
floatdef: "float"! ID (ASSIGN! expr {#floatdef = #([FLOATDEF,"FLOATDEF"],
floatdef);} | (COMMA! ID)* {#floatdef = #([EMPTYFLOATDEF,"EMPTYFLOATDEF"],
floatdef);});
```

```
/*****************************************
 * 5.2 OSCILLATOR DEFINITION
 *****************************************/

oscdef : "osc"! ID (ASSIGN! STRING {#oscdef = #([OSCDEF,"OSCDEF"], oscdef);}
                                  | (COMMA! ID)* {#oscdef =
#([EMPTYOSCDEF,"EMPTYOSCDEF"], oscdef);});

/*****************************************
 * 5.3 OSCILLATOR BANK DEFINITION
 *****************************************/
oscbdef : "oscbank"! ID ASSIGN! expr {#oscbdef = #([OSCBANKDEF,"OSCBANKDEF"],
oscbdef);};
oscbelementdef : ID LBRACK! (ID | INT) RBRACK! ASSIGN! STRING
                                  {#oscbelementdef =
#([OSCBANKELEMENT_DEF,"OSCBANKELEMENT_DEF"], oscbelementdef);};
oscBank : ID LBRACK! (expr) RBRACK! {#oscBank = #([OSCBANK,"OSCBANK"],
oscBank);};

/*****************************************
 * 5.4 ENVELOPE DEFINITION
 *****************************************/
envdef : "env"! ID ( ASSIGN! LBRACE! (LPAREN! FLOAT COMMA! FLOAT RPAREN!)+
RBRACE!
                                         {#envdef = #([ENVDEF,"ENVDEF"],
envdef);}
                                  | (COMMA! ID)* {#envdef =
#([EMPTYENVDEF,"EMPTYENVDEF"], envdef);});

/*****************************************
 * 5.5 MIXER DEFINITION
 *****************************************/
mixdef : "mixer"! ID (COMMA! ID)* {#mixdef = #([MIXERDEF,"MIXERDEF"],
mixdef);};

/* ****************************************
 * 6 STATEMENTS
 * NOTE: Statements are NOT universal.
 * Within each section (Definition/Connection),
 * DIFFERENT statements are allowed
 * ****************************************/

/*****************************************
 * 6.1 CONDITIONAL STATEMENTS
 *****************************************/
//[DEFINITION SECTION]
conditionaldef  : ifdef
                           | fordef
                           ;

//[CONNECTION SECTION]
conditionalcon : ifcon
                       | forcon
                       ;

/*****************************************
 * 6.1.1 FOR LOOPS
 *****************************************/
//[DEFINITION SECTION]
fordef : ( FOR^ LPAREN! assignment SEMIC! bool SEMIC! delta RPAREN! LBRACE!
defstatements RBRACE! )
```

```
;

//[CONNECTION SECTION]
forcon : "for"^ LPAREN!  assignment SEMIC! bool SEMIC! delta RPAREN! LBRACE!
  constatements RBRACE!;

/*****************************************
 * 6.1.2 IF/ELSE STATEMENTS
 *****************************************/
//[DEFINITION SECTION]
ifdef : ( IF^ LPAREN! (bool) RPAREN! LBRACE! defstatements RBRACE!
  (options {greedy=true;}: "else"! LBRACE! defstatements RBRACE!)? )
;

//[CONNECTION SECTION]
// testConditionalCon()
ifcon : IF^ LPAREN! bool RPAREN! LBRACE! constatements RBRACE!
               (options {greedy=true;} : "else"! LBRACE! constatements
RBRACE!)?
               (EOF!)?; // TODO Needed by test case, check if affects program

// testDelta()
delta : assignment
| ID INC^
| ID DEC^;

/*****************************************
 * 6.2 CONNECTION STATEMENTS
 *****************************************/
//[CONNECTION SECTION]
constatements: (osccon | mixstmt | conditionalcon)*
       {#constatements = #([CONSTATEMENTS,"CONSTATEMENTS"], constatements);};

/*****************************************
 * 6.2.1 OSCILLATOR CONNECTIONS
 *****************************************/
//[CONNECTION SECTION]
bankIndex : LBRACK! expr RBRACK! {#bankIndex = #([BANKINDEX, "BANKINDEX"],
bankIndex);}
                    | {#bankIndex = #([BANKINDEX, "BANKINDEX"], bankIndex);};
value : INT | FLOAT;
cfreq : AT! expr;
oscconargs1 : (expr | oscBank) (cfreq|EOF!)?
                    {#oscconargs1 = #([OSCCONARG1, "OSCCONARG1"],
oscconargs1);}
                    | funccall (cfreq)?
                    {#oscconargs1 = #([OSCCONARG1, "OSCCONARG1"],
oscconargs1);}
                    ;
oscconargs2 : (expr | oscBank)
                    {#oscconargs2 = #([OSCCONARG2, "OSCCONARG2"],
oscconargs2);}
                    | funccall
                    {#oscconargs2 = #([OSCCONARG2, "OSCCONARG2"],
oscconargs2);}
                    ;
osccon : ID bankIndex LPAREN! oscconargs1 COMMA! oscconargs2 RPAREN! SEMIC!
              {#osccon = #([OSCCON,"OSCCON"], osccon);}
              ;


/*****************************************
 * 6.2.2 MIXER CONNECTIONS
 *****************************************/
// testMixStmt()
mixstmt      :      ID ASSIGN! mixcon SEMIC!
                          {#mixstmt = #([MIXSTMT, "MIXSTMT"], mixstmt);};

//This rule is currently pretty useless.
mixcon : mixterms;
mixterms : mixterm (PLUS^ mixterms)? (options{greedy=true;}:EOF!)?;
mixterm : (FLOAT | INT | funccall | ID | oscBank) (TIMES^ mixterm)?
(options{greedy=true;}:EOF!)?;
//In general, check for EOFs when using ?'s. {greedy=true;} prob not necessary
//Else the tests will fail, at least.

/******************************************
 * 6.3 OUTPUT EQUATION
 *****************************************/
// See 7.3? But mixstmts don't have expressions, whereas 7.3 can?

/* ***************************************
 * 7 FUNCTIONS
 *****************************************/
// testFuncDef()
funcdef : START! FUNC! ID LPAREN! arg_list_decl RPAREN! defsec consec outputeq
END! FUNC! ID!
                          {#funcdef = #([FUNCDEF, "FUNCDEF"], funcdef);};
arg_list_decl : data_type ID (COMMA! data_type ID)*
                          {#arg_list_decl = #([ARG_LIST_DECL,
"ARG_LIST_DECLARATION"], arg_list_decl);}
                          | "void"^
                          {#arg_list_decl = #([ARG_LIST_DECL,
"ARG_LIST_DECLARATION"], arg_list_decl);};
data_type : "mixer"^
                    |"env"^
                    |"osc"^
                    |"oscbank"^
                    |"int"^
                    |"float"^;

/* NOT USED!
arg_list : (INT | FLOAT | ID) (COMMA! (INT| FLOAT | ID))*
                    {#arg_list = #([ARG_LIST, "ARG_LIST"], arg_list);}
                    | "void"^
                    {#arg_list = #([ARG_LIST, "ARG_LIST"], arg_list);}
                    ;
*/
funccall : ID LBRACE! expr_list RBRACE!
                    { #funccall = #([FUNCCALL,"FUNCCALL"], #funccall); };

// testExprList()
expr_list  : expr (COMMA! expr)*
                    {#expr_list = #([EXPR_LIST,"EXPR_LIST"], expr_list);}
                    |/*nothing*/
                    {#expr_list = #([EXPR_LIST,"EXPR_LIST"], expr_list);}
                    ;

/******************************************
 * 7.1 DEFINITION SECTION
 *****************************************/
// testDefSection()
defsec : START! DEF! defstatements END! DEF!;

/******************************************
 * 7.2 CONNECTION SECTION
 *****************************************/
// testConnectSection()
consec : START! CONNECT!  constatements  END! CONNECT!;

/******************************************
 * 7.3 RETURN VALUE
 *****************************************/
// testOutputEquation()
outputeq : OUTPUT^ ASSIGN! mixcon SEMIC!
;

/******************************************
 * 9 AASL PROGRAM
 *****************************************/
program :  head (funcdef)* main
       {#program = #([PROGRAM,"PROGRAM"], program);}
;

/******************************************
 * 9.1 HEADER SECTION
 *****************************************/
head : "start"! "header"! outset lenset (segset)? "end"! "header"!
       {#head = #([HEAD, "HEAD"], head);}
;

outset : OUTPUT^ ASSIGN! STRING
;

lenset : TONELENGTH^ ASSIGN! (FLOAT | INT)
;

segset : SEGMENTS^ ASSIGN! INT
;

/******************************************
 * 9.2 MAIN FUNCTION
 *****************************************/
main : START! FUNC! MAIN^ defsec consec outputeq END! FUNC! MAIN!
;


/************************************
 * 10 Sandbox
 ***********************************/
// test : INT INT^ INT INT INT;
//test : INT (expr)^ INT INT INT;
//expr does NOT return a root node. Well, rules can't be nodes, so...
//Notes


//Scrap
//Matches: "arg1, arg2"
//Reference: LRM 6.2.1 Oscillator Connections
//This would be one way
/*
oscargs : (INT
                |FLOAT
                |((ID (bankIndex)?)|funccall) LPAREN!(FLOAT|INT)RPAREN!)
                )
                COMMA!
                (FLOAT
                |ID (bankIndex)?
                |funccall
```

```
               );
*/
/*
osccon : (INT
               | FLOAT
               | (( (ID (LBRACK! (INT | ID) RBRACK!)? )
                    | funccall)
                   LPAREN! (FLOAT | INT) RPAREN!)
        )
            COMMA!
               (FLOAT
               | ID (LBRACK! (INT | ID) RBRACK!)?
               | funccall);
*/


/*
 * ==================================================================
 *                                                  LEXER
 * ==================================================================
 */
class AASLLexer extends Lexer;
options
{
testLiterals = false;
k = 3;
charVocabulary = '\3'..'\377';
}

{
    public int nr_error = 0;
    public void reportError( String s ) {
        super.reportError( s );
        nr_error++;
    }
    public void reportError( RecognitionException e ) {
        super.reportError( e );
        nr_error++;
    }
}

PLUS : '+' ;
MINUS : '-' ;
TIMES : '*' ;
DIV : '/' ;
MOD : '%' ;
ASSIGN : '=' ;
NOT : '!' ;
LT : '<' ;
GT : '>' ;
LE : "<=" ;
GE : ">=" ;
EQ : "==" ;
NE : "!=";
AND : "&&" ;
OR : "||" ;
INC : "++" ;
DEC : "--" ;
COMMA   : ',' ;
SEMIC   : ';' ;
DQUOTE  : '"' ;
LPAREN  : '(' ;
RPAREN  : ')' ;
```

```
LBRACE  : '{' ;
RBRACE  : '}' ;
LBRACK : '[';
RBRACK : ']';
AT : '@';

/*
 * The protected keyword here just means that this
 * production can be used only internally in the lexer.
 * The lexer will never return a DIGIT token.
 */
protected LETTER : ('a'..'z' | 'A'..'Z') ;
protected DIGIT : '0'..'9' ;
protected INT : ('0'..'9')+;
protected FLOAT : INT '.' INT;

/*
 * We will be defining keywords within our parser,
 * which will end up generating tokens called LITERAL_begin
 * and LITERAL_end, for example.  Recall that these literals
 * will be inserted in a special hash table.
 *
 * testLiterals=true generates code that will
 * lookup a name in the literals hash table if the IDENT rule
 * matches a word. This implies that every literal you specify
 * in the parser grammar must be matchable by the IDENT rule.
 * If a word such as "end" is found in the literals table,
 * the token type will be changed to be something like LITERAL_end
 * before being returned to the parser.
 */
ID options {testLiterals = true; }
  : (LETTER | '_') (LETTER | DIGIT | '_')* ;

NUMBER : (DIGIT)+
        ( '.' (DIGIT)* {$setType(FLOAT);}
        |              {$setType(INT);}
        )
;

STRING : '"'! ('"' '"'! | ~('"'))* '"'! ;

WS : ( ' ' | '\t' ) { $setType(Token.SKIP); } ;
NEWLINE : ('\n' | ('\r' '\n') => '\r' '\n' | '\r') { $setType(Token.SKIP);
newline(); } ;

//COMMENTS : ('/' '/' (~('\n'|'\r'))* ) {$setType(Token.SKIP);}| ('/' '*' (
LETTER | ~('a'..'z'|'A'..'Z') )* '*' '/') {$setType(Token.SKIP);};
//COMMENTS : ('/' '/' (~('\n'|'\r'))* ) {$setType(Token.SKIP);}
//           | ("/*" ( LETTER | ~('a'..'z'|'A'..'Z') )* "*/")
{$setType(Token.SKIP);};
SL_COMMENT
     :    "//" (~('\n'|'\r'))*
         {$setType(Token.SKIP);}

     ;
COMMENTS     :    "/*"
         (
           options {
                 generateAmbigWarnings=false;
             }
         :
             { LA(2)!='/' }? '*'               // match * as long as its not
followed by /
```

```
        |    '\r' '\n'        {newline();}
        |    '\r'            {newline();}
        |    '\n'            {newline();}
        |    ~('*'|'\n'|'\r')
        )*
        "*/"
        {$setType(Token.SKIP);}
    ;
```

walker.g

```
/*
 * Syntactic Check
 * by Rob Katz, Carlos R. Perez, Vaishnav Janardhan
 *
 */

header {
package aasl;

import java.io.*;
import java.util.*;
import aasl.intermediateTypes.*;
}

class AaslAntlrWalker extends TreeParser;
options
{
      importVocab = AASLParser;
}

{
      public Vector v = new Vector();
      public Vector funcV = new Vector();
      public AaslSymbolTable symt = new AaslSymbolTable(null);
      public AaslSegArray seg_arr;
      public float centralFreq=0;
      public int outputNumber=0;
      public AaslSegArray seg_arr_temp;
      boolean output_bool=false;
      String str;

      public String javaOutputFile;
      public String line_stat;

      public String lineStats(AST cur) {
            if (cur == null)
                  return "--";
            String ret = cur.getLine() + ":" + cur.getColumn();
            line_stat = ret;
            return ret;
      }
}

expr returns [AaslType r]
```

```
{
      AaslType a,b,c;
      AaslType[] x;
      r=null;
}
            : #(PROGRAM (a=expr)* )
            | #(MAIN a=expr b=expr c=expr)
            | #(HEAD a=expr b=expr (c=expr)?)
            | #(OR {String linenr=lineStats(_t);} a=expr b=expr)
            {
                  /* check for errors in argument types */
                  if (!(a instanceof AaslSegArray || a instanceof
AaslBool) &&
                        !(b instanceof AaslSegArray || b instanceof
AaslBool))
                        throw new AaslException(linenr, "Semantic
Error: expecting "+
                        "AaslSegArray or AaslBool in AND
subexpressions.");

                  /*
                   * #OR Rule is the only one statement that must MERGE
for multiple
                   * SEG statements. So if a&&b == AaslSegArray, merge
the SegArrays.
                   */
                  if (a instanceof AaslSegArray && b instanceof
AaslSegArray) {
                        AaslSegArray merged = (AaslSegArray) a;
                        merged.merge( (AaslSegArray) b );
                        r = merged; /* #IF will make this the default
seg_arr */
                  }
                  else

                  /* return AaslBool if both types are boolean */
                  if (a instanceof AaslBool && b instanceof AaslBool) {
                        if ( ((AaslBool)a).value || ((AaslBool)b).value
)     //changed!!!
                              r = new AaslBool(true); /* true (any of
them) */ //changed!!!
                        else
                              r = new AaslBool(false);
                  }
                  else {
                        /* return AaslSegArray if one type is of
AaslSegArray, other MUST be AaslBool */
                        if (a instanceof AaslBool) {
                              r=b;
                        } else {
                              r=a;
                        }
                  }
            }
            | #(AND {String linenr=lineStats(_t);} a=expr b=expr)
            {
                  /* check for errors in argument types */
                  if (!(a instanceof AaslSegArray || a instanceof
AaslBool) &&
                        !(b instanceof AaslSegArray || b instanceof
AaslBool))
                        throw new AaslException(linenr, "Semantic
Error: expecting "+
                        "AaslSegArray or AaslBool in AND
subexpressions.");

                  /*
                   * #AND Rule is the only one statement that must
check for multiple
                   * SEG statements. So if a&&b == AaslSegArray, throw
Semantic Error
                   */
                  if (a instanceof AaslSegArray && b instanceof
AaslSegArray)
                        throw new AaslException(linenr,"Semantic Error:
Two SEGments "+
                        "found in AND statements!" );
                  else

                  /* return AaslBool if both types are boolean */
                  if (a instanceof AaslBool && b instanceof AaslBool) {
                        if ( ((AaslBool)a).value && ((AaslBool)b).value
)
                              r = a; /* true */
                        else
                              r = new AaslBool(false);
                  }
                  else {
                        /* return AaslSegArray if one type is of
AaslSegArray, other MUST be AaslBool */
                        if (a instanceof AaslBool) {
                              if ( ((AaslBool)a).value )
                                    r = b; /* bool==true THEN return
AaslSegArray */
                              else
                                    r = new AaslBool(false); /*
bool==false THEN don't send AaslSegArray*/
                        } else {
                              /* b MUST be of AaslBool and a MUST be
AaslSegArray */
                              if ( ((AaslBool)b).value )
                                    r = a; /* bool==true THEN return
AaslSegArray */
                              else
                                    r = new AaslBool(false); /*
bool==false THEN don't send AaslSegArray*/
                        }
                  }
            }
            | #(SEG {String linenr=lineStats(_t);} a=expr)
            {
                  /*
                   * Returns AaslSegArray
                   */
                  if (a instanceof AaslInt) {
                        AaslSegArray segArray = new
AaslSegArray(seg_arr.length,seg_arr);
                        try{ segArray.setValue(AaslType.intValue(a)); }
                        catch (ArrayIndexOutOfBoundsException e) {
                              throw new AaslException(linenr,"SEGment
assignemt 'must' be in defined range[0-"+segArray.length+"]!");
                        }
                        r = segArray;
                  } else
                        throw new AaslException(linenr, "SEGment
assignment 'must' be INTEGER type.");
            }
            | string:STRING
            {
                  AaslString str = new AaslString( string.getText() );
                  r = str;
            }
            | #(TONELENGTH a=expr)
            {
                  v.add(1, new AaslFloat(AaslType.floatValue(a)));
                  r=a;

                  //might as well set segments to 1 here since seg
statement is optional
                   v.add(2, new AaslInt(1) );
                   seg_arr = new AaslSegArray(1, null);
                   seg_arr.allTrue();
            }
            | val:FLOAT       {r = new AaslFloat(
Float.valueOf(val.getText()).floatValue() );}
            | val2:INT        {r = new AaslInt(
Integer.parseInt(val2.getText()) );}
            | rT:TRUE         {r = new AaslBool(true);}
            | rF:FALSE        {r = new AaslBool(false);}
            | id:ID
            {
                  if (symt.getValue(id.getText())==null)
                        r = new AaslString(id.getText());
                  else
                        r = symt.getValue(id.getText());
            }
            | #(OSCBANK {String linenr=lineStats(_t);} bankName:ID
b=expr)
            {
                  if(b instanceof AaslString)
                        throw new AaslException(linenr,"Bad index given
for oscBank " + bankName.getText());
                  String name = bankName.getText() + "$" +
AaslType.intValue(b);
                  if(symt.containsSymbol(name))
                        r = symt.getValue(name);
                  else
                        throw new AaslException(linenr,"Oscillator bank
element "+bankName.getText()
                        + "[" + AaslType.intValue(b) + "] not
defined.");
```

```
            }
            | #(ASSIGN {String linenr=lineStats(_t); AaslType rvalue;}
lvalue:ID rvalue=expr )
            {
                  if(!symt.containsSymbol(lvalue.getText()) )
                        throw new AaslException(linenr,lvalue.getText()
+ " not declared.");
                  AaslType type = symt.getValue(lvalue.getText());
                  if(type instanceof AaslFloat || type instanceof
AaslInt)
                        symt.setValue( lvalue.getText(), rvalue );
                  else if(type instanceof AaslOscillator2){
                        v.remove(type);

      ((AaslOscillator2)type).setWaveType((AaslString)rvalue,seg_arr);
                        v.add(type);
                        symt.setValue(lvalue.getText(), type);
                  }
                  else
                        throw new AaslException(linenr,"Illegal
Assignment");
            }
            | #(SEGMENTS {String linenr=lineStats(_t);} a=expr)
            {
                  if (a instanceof AaslInt) {
                        seg_arr = new AaslSegArray(
((AaslInt)a).getValue(), null);
                        seg_arr.allTrue();
                        seg_arr_temp =  new AaslSegArray(
((AaslInt)a).getValue(), null);
                        v.remove(2);
                        v.add(2, a);
                        r=a;
                  }
                  else
                        a.error(linenr,"Segment Assignment.");
            }
            | #(NOT a=expr)   {r = a.not();}
            | #(EQ a=expr b=expr)   {r = a.eq(b);}
            | #(NE a=expr b=expr)   {r = a.ne(b);}
            | #(LT a=expr b=expr)   {r = a.lt(b);}
            | #(LE a=expr b=expr)   {r = a.le(b);}
            | #(GT a=expr b=expr)   {r = a.gt(b);}
            | #(GE a=expr b=expr)   {r = a.ge(b);}
            | #(PLUS a=expr b=expr) {r = a.plus(b, seg_arr, v);}
            | #(MINUS a=expr b=expr) {r = a.minus(b, seg_arr);}
            | #(TIMES a=expr b=expr) {r = a.times(b, seg_arr);}
            | #(DIV a=expr b=expr)  {
                  float rval = AaslType.floatValue(b);
                  if (!(rval > (float)0.0 || rval < (float)0.0))
                        a.error("Divide by zero!");
                  r = a.div(b, seg_arr);}

/* ***************************************
 * 5 DEFINITIONS
 ****************************************/
            | #(DEFSTATEMENTS (a=expr)*)
            | #(INTDEF {String linenr=lineStats(_t);} intName:ID
a=expr)
            {
                  if (symt.topContainsSymbol(intName.getText()))
                              throw new AaslException(linenr,
intName.getText()+" exists! Cannot be redefined!");
                  symt.add(intName.getText(), new
AaslInt(AaslType.intValue(a)));
            }
            | #(FLOATDEF {String linenr=lineStats(_t);} floatName:ID
a=expr)
            {
                  if (symt.topContainsSymbol(floatName.getText()))
                              throw new AaslException(linenr,
floatName.getText()+" exists! Cannot be redefined!");
                  symt.add(floatName.getText(), new
AaslFloat(AaslType.floatValue(a)));
            }
            | #(OSCDEF {String linenr=lineStats(_t);} oscName:ID
b=expr)
            {
                  /*
                   * Expecting (OSCDEF ID
("SINE"|"SQUARE"|"SAW"|"REVSAW") )
                   */
                  if (!(b instanceof AaslString))
                        b.error(linenr,"Osc Definition");

                  /*
                   * Can we redefine? no
                   */
                  if (symt.topContainsSymbol(oscName.getText()))
                              throw new
AaslException(linenr,"Oscillator
<"+oscName.getText()+":"+oscName.getLine()+"> already defined in
scope!");
                  else
                  {
                        //Call the AaslOscillator2 constructor, passing
it a.value as the name, b.value
                        //    as the waveType, seg_arr.length as the
number of segments, and seg_arr as the
                        //    AaslSegArray.  Add this new object to the
vector and add an entry to the current
                        //    symbol table with a.value as the key and
AaslOscillator2 as the type.
                        AaslOscillator2 new_osc = new
AaslOscillator2(oscName.getText(),((AaslString)b).value,seg_arr.length,
seg_arr);
                        symt.put(oscName.getText(),new_osc);
                        v.add(new_osc);
                        r = new_osc;
                  }
            }     ///////Have to update Mixers and OSC banks to make
them compatible with vectors
            | #(OSCBANKDEF {String linenr=lineStats(_t);} oscbName:ID
b=expr)
```

```
            {
                        if (!(b instanceof AaslInt))//AaslString))
                              b.error(linenr,"OscBank Definition");

                  /*
                   * Can we redefine? NO.
                   */
                        if (symt.topContainsSymbol(oscbName.getText()))
                              throw new AaslException(linenr,"OscBank
<"+ oscbName.getText()+":"+oscbName.getLine()+"> already defined in
scope!");
                        else
                        {
                              for(int i=0; i<((AaslInt)b).getValue();
i++){
                                    AaslOscillator2 new_osc = new
AaslOscillator2(oscbName.getText()+"$"+i, "SINE", seg_arr.length,
seg_arr);
                                    symt.put(new_osc.name,new_osc);
                                    v.add(new_osc);
                              }
                        }
            }
            |#(OSCBANKELEMENT_DEF {String linenr=lineStats(_t);}
oscBankName:ID {AaslType bankNum;} bankNum=expr oscType:STRING)
            { /* ID (ID | INT)  ("SINE"|"SQUARE"|"SAW"|"REVSAW") */
                  String oscBankName_String =
oscBankName.getText()+"$"+AaslType.intValue(bankNum);

                  AaslType this_type =
symt.getValue(oscBankName_String);
                  if(this_type == null)
                        throw new
AaslException(linenr,oscBankName.getText()+"["+AaslType.intValue(bankNu
m)+"] is not defined!");

                  if(this_type.getType().compareTo("oscillator")!=0)
                        throw new AaslException(linenr,"This ain't no
oscillator, dummy! Rob don't like it!");
                  AaslOscillator2 this_osc =
(AaslOscillator2)this_type;
                  if(this_osc == null)
                        throw new AaslException(linenr,"OscBank <"+
oscBankName.getText()+":"+oscBankName.getLine()+"> not defined.");
                  v.remove(this_osc);
                  this_osc.setWaveType(new
AaslString(oscType.getText()),seg_arr);
                  v.add(this_osc);
            }
            | #(ENVDEF {String linenr=lineStats(_t);} envName:ID
floatTuple:. ) /* ("ENVDEF" ID (FLOAT FLOAT)+ ) */
            {
                  /* Can't redefine ENVELOPE, if it already exists in
the CURRENT scope only */
                  if (symt.topContainsSymbol(envName.getText()))
```

```
                        throw new AaslException(linenr,"Envelope
<"+envName.getText()+":"+envName.getLine()+"> already defined in
scope!");

                  /* Checks that the tuple is all of type float */
                  for (AST floatType=#floatTuple; floatType!=null;
floatType=floatType.getNextSibling()) {
                        match(floatType,FLOAT);
                  }
                  AaslEnvelope env = new
AaslEnvelope(envName.getText(),#floatTuple);

                  symt.add(envName.getText(),env);
                  v.add(env);
                  r = env;
            }
            | #(MIXERDEF {String linenr=lineStats(_t);} mixerDefName:.)
            {
                  for (AST cur=#mixerDefName; cur!=null;
cur=cur.getNextSibling()) {
                        /* Can't redefine ENVELOPE, if it already
exists in the CURRENT scope only */
                        if (symt.topContainsSymbol(cur.getText()))
                              throw new AaslException(linenr,
cur.getText()+" exists! Cannot be redefined!");
                        else
                        {
                              AaslMixer2 new_mixer = new
AaslMixer2(cur.getText(),seg_arr.length);
                              symt.put(cur.getText(),new_mixer);
                              v.add(new_mixer);
                              r = new_mixer;
                        }
                  }
            }

/* ***************************************
 * 5.B EMPTY DEFINITIONS
 ****************************************/
            | #(EMPTYOSCDEF {String linenr=lineStats(_t);}
oscDefName:.)
            {
                  for (AST cur=#oscDefName; cur!=null;
cur=cur.getNextSibling()) {
                        /* Can't redefine ENVELOPE, if it already
exists in the CURRENT scope only */
                        if (symt.topContainsSymbol(cur.getText()))
                              throw new AaslException(linenr,
cur.getText()+" exists! Cannot be redefined!");

                        AaslOscillator2 new_osc = new
AaslOscillator2(cur.getText(),"SINE",seg_arr.length,seg_arr);
                        symt.put(cur.getText(),new_osc);
                        v.add(new_osc);
                        r = new_osc;

                  }
            }
            | #(EMPTYENVDEF {String linenr=lineStats(_t);}
envDefName:.)
            {
                  for (AST cur=#envDefName; cur!=null;
cur=cur.getNextSibling()) {
                        /* Can't redefine ENVELOPE, if it already
exists in the CURRENT scope only */
                        if (symt.topContainsSymbol(cur.getText()))
                              throw new AaslException(linenr,
cur.getText()+" exists! Cannot be redefined!");

                        AaslEnvelope env = new
AaslEnvelope(cur.getText());
                        symt.add(cur.getText(),env);
                        v.add(env);
                        r = env;
                  }
            }
            | #(EMPTYINTDEF {String linenr=lineStats(_t);}
intDefName:.)
            {
                  for (AST cur=#intDefName; cur!=null;
cur=cur.getNextSibling()) {
                        if (symt.topContainsSymbol(cur.getText()))
                              throw new
AaslException(linenr,cur.getText()+" exists! Cannot be redefined!");
                        AaslInt new_int = new AaslInt(0);
                        symt.add(cur.getText(), new_int);
                        r = new_int;
                  }
            }
            | #(EMPTYFLOATDEF {String linenr=lineStats(_t);}
floatDefName:.)
            {
                  for (AST cur=#floatDefName; cur!=null;
cur=cur.getNextSibling()) {
                        if (symt.topContainsSymbol(cur.getText()))
                              throw new
AaslException(linenr,cur.getText()+" exists! Cannot be redefined!");
                        AaslFloat new_float = new
AaslFloat((float)0.0);
                        symt.add(cur.getText(), new_float);
                        r = new_float;
                  }
            }


/* ***************************************
 * 6.2 CONNECTION STATEMENTS
 ****************************************/
            | #(CONSTATEMENTS (a=expr)* )
            | #(BANKINDEX {a=null;}(a=expr)? )
            {
                  r=a;
            }
            | #(OSCCON {String linenr=lineStats(_t);} oscname:ID a=expr
b=expr c=expr)
            {
                  String osc_name = oscname.getText();
                  if(a!=null) //if this is an oscillator bank...
                        osc_name += "$" + AaslType.intValue(a);
      //...convert name
                  if(!symt.containsSymbol(osc_name))  //if not
defined...
                        throw new AaslException(linenr,osc_name + " not
defined.");
                  if(!(symt.getValue(osc_name) instanceof
AaslOscillator2)) //if not an oscillator
                        throw new AaslException(linenr,osc_name + "
being treated as an oscillator.");
                  if(!(AaslType.checkFloatRange((AaslType)c)))
                  {
                        if( c instanceof AaslFloat)
                              throw new AaslException(linenr,"The
amplitude value = "+((AaslFloat)c).value+" is out of range");
                        if( c instanceof AaslInt)
                              throw new AaslException(linenr,"The
amplitude value = "+((AaslInt)c).value+" is out of range");

                  }
                  AaslOscillator2 this_osc =
(AaslOscillator2)(symt.getValue(osc_name));
                  v.remove(this_osc);
                  for(int i=0; i<seg_arr.length; i++)
                  {
                        if(seg_arr.seg_arr[i]){
                              this_osc.freq[i]=b;
                              this_osc.amp[i]=c;
                              this_osc.centralFreq[i]=centralFreq;
                        }
                  }
                  v.add(this_osc);
                  symt.setValue(osc_name, this_osc);
            }
            | #(OSCCONARG1 {String linenr=lineStats(_t);} a=expr
{c=null;}(c=expr)?)
            {
                  if(a instanceof AaslFloat || a instanceof AaslInt)
                        r=a;
                  else if(c==null)
                        throw new AaslException(linenr,"Central
frequency missing in oscillator freq connection");
                  else
                  {
                        String con_name = a.name;
                        if(!symt.containsSymbol(con_name))
                              throw new AaslException(linenr,con_name +
" is not declared.");
                        centralFreq=AaslType.floatValue(c);
                        r=symt.getValue(con_name);
                  }
            }
            | #(OSCCONARG2 {String linenr=lineStats(_t);} a=expr)
            {
                  //System.out.println(a.getType());
                  if(!(a instanceof AaslString))
                        r = a;
                  else
                  {
                        String con_name;
                        con_name = ((AaslString)a).value;
                        if(!symt.containsSymbol(con_name))
                              throw new AaslException(linenr,con_name +
" is not declared.");
                        r=symt.getValue(con_name);
                  }
                  if(    (r instanceof AaslInt || r instanceof
AaslFloat) &&
                         (AaslType.floatValue(r)>1.0 ||
AaslType.floatValue(r)<0.0) )
                  {
                        if( r instanceof AaslFloat)
                              throw new AaslException(linenr,"The
amplitude value = "+((AaslFloat)r).value+" is out of range");
                        if( r instanceof AaslInt)
                              throw new AaslException(linenr,"The
amplitude value = "+((AaslInt)r).value+" is out of range");

                  }
            }
            | #(MIXSTMT {String linenr=lineStats(_t);} mixName:ID
b=expr)
            {
                  if(!symt.containsSymbol(mixName.getText()))
                        throw new AaslException(linenr,"Mixer " +
mixName.getText() + " not defined.");
                  AaslType type = symt.getValue(mixName.getText());
                  if(!(type instanceof AaslMixer2)){
                        symt.setValue(mixName.getText(),b);
                  }
                  if(b instanceof AaslMixer2){
                        b.name = mixName.getText();
                        symt.setValue(mixName.getText(), b);
                  }
                  else if(b instanceof AaslOscillator2){
                        AaslMixer2 mix_remove =
(AaslMixer2)symt.getValue(mixName.getText());
                        v.remove(mix_remove);
                        AaslMixer2 mix = new
AaslMixer2(mixName.getText(), seg_arr.length);
                        mix.addElement(b);
                        symt.setValue(mixName.getText(), mix);
                        v.add(mix);
                  }
                  else if(b instanceof AaslExecutedFunction){
                        AaslMixer2 mix = new
AaslMixer2(mixName.getText(), seg_arr.length);
                        mix.addElement(b);
                        symt.setValue(mixName.getText(), mix);
                        v.add(mix);
                }
        }
        | #("if"
                        {
                        String linenr=lineStats(_t);
                        seg_arr = new
AaslSegArray(seg_arr.get_length(), seg_arr);
                        symt = new AaslSymbolTable(symt);
                        }
        a=expr thenp:. (elsep:.)?)
        {
                /* check for errors in argument types */
                if (!(a instanceof AaslSegArray || a instanceof
AaslBool))
                        throw new AaslException(linenr, "Semantic
Error: expecting "+
                        "AaslSegArray or AaslBool in IF expressions,
got ."+a.getType());

                /* expression IS AaslBool */
                if (a instanceof AaslBool) {
                        if ( ((AaslBool)a).value ) { /* true */
                                r = expr(#thenp);
                        } else {
                                seg_arr.allFalse(); /* no particular
reason, just make sure its false */
                                if (#elsep != null) {
                                        /* ONLY run if #elsep has been
defined, otherwise just do nothing */
                                        seg_arr.invert();
                                        r = expr(#elsep);
                                }
                        }
                } else {
                        /* expression IS AaslSegArray, THEN the
expression is TRUE */
                        AaslSegArray old_seg_arr=seg_arr;
                        seg_arr=(AaslSegArray)a;
                        r = expr(#thenp);
                        if(#elsep != null){
                                seg_arr.invert();
                                r = expr(#elsep);
                        }
                        seg_arr=old_seg_arr;
                }


                /*
                 * Restore SegmentArray and SymbolTable after end of
scope
                 */
                seg_arr = seg_arr.parent;
                symt = symt.parent;
        }
        | #("for" {String linenr=lineStats(_t);} assignment:.
bool:. delta:. for_body:. )
```

```
            {
                  ForLoopInstruction for_loop = new
ForLoopInstruction(this,#assignment,#bool,#delta,#for_body);
                  r = for_loop.run();
            }


/* ***************************************
 * 7 FUNCTIONS
 ****************************************/
            | #(FUNCCALL {String linenr=lineStats(_t);} fcname:ID
{AaslFunctionCallArguments fargs;} fargs=arg_list)
            {
                  /*
                   * Support function overloading <polymorphism> based
on func args.
                   * and Dynamic Scoping.
                   *
                   * Whenever this is called, we replace the this
object with the
                   * result (func eval).
                   *
                   * Constructor( <vector> of function definitions,
                   *          <SymbolTable> of previous scope used for
argument substitution,
                   *          <String> function name,
<AaslFunctionCallArguments>
                   *
                   * Error checks done: function is defined, function
params match,
                   *          farguments are defined
                   */
                  AaslFunctionCall fcall = new AaslFunctionCall(funcV,
symt, fcname.getText(), fargs);
                  AaslSegArray old_seg_array = seg_arr;
                  seg_arr = new AaslSegArray(seg_arr.length, null);
                  seg_arr.allTrue();
                  r = fcall.evaluate(this); //Dynamic Scoping
                  seg_arr = old_seg_array;
            }
            | #(FUNCDEF {String linenr=lineStats(_t);} fdefname:ID
{AaslFunctionDefArgList fdefargs;} fdefargs=arg_list_decl fbodyDef:.
fbodyCon:. fbodyOut:.)
            {
                  AaslFunctionDef fdef = new AaslFunctionDef();
                  fdef.setFunctionName( fdefname.getText() );
                  fdef.setFunctionPrototype( fdefargs );
                  fdef.setASTFunctionBody( #fbodyDef, #fbodyCon,
#fbodyOut );
                  /* Add this func decl to funcVector */
                  funcV.add( fdef );

                  r = fdef;
            }
            | #(OUTPUT {String linenr=lineStats(_t);} a=expr)
            {
                  if(a instanceof AaslString && output_bool==false){
                        v.add(0, a);
                        this.javaOutputFile = ((AaslString)a).value;
                        output_bool=true;
                  }
                  else if(a instanceof AaslString){
                        throw new AaslException(linenr, "Illegal output
assignment: " + ((AaslString)a).value);
                  }
                  else{
                        if(!(a instanceof AaslMixer2 || a instanceof
AaslOscillator2))
                              throw new AaslException(linenr,"Attempted
to assign output to something other than a" +
                                    "mixer or an oscillator! " +
((AaslType)a).getType());
                        if(a instanceof AaslMixer2){
                              ((AaslMixer2)a).name="output$$" +
++outputNumber;
                              ((AaslMixer2)a).setAsOutput(seg_arr);
                              symt.put(a.name, a);
                              v.add(a);
                        }
                        else if(a instanceof AaslOscillator2){
                              AaslMixer2 this_mix = new
AaslMixer2("output$$" + ++outputNumber, seg_arr.length);
                              this_mix.addElement(a);
                              this_mix.setAsOutput(seg_arr);
                              symt.put(this_mix.name, this_mix);
                              v.add(this_mix);
                        }
                  }
            }
            ;

arg_list returns [AaslFunctionCallArguments r] {r=null;}
            : #(EXPR_LIST argListBody:.)
            {
                  r = new AaslFunctionCallArguments(#argListBody);
            };

arg_list_decl returns [AaslFunctionDefArgList r] {r=null;}
            : #(ARG_LIST_DECL argListDecl:. )
            {
                  r = new AaslFunctionDefArgList(#argListDecl);
            };
```

AaslType.java

```
/*
 * AaslType.java
 * Written by Rob Katz, Carlos R. Perez, Vaishnav Janardhan
 *
 */
```

```
package aasl.intermediateTypes;

import java.util.*;
import aasl.*;

public class AaslType {
      public String name;

      public AaslType(){
            name=null;
      }

      public AaslType(String name){
            this.name=name;
      }

//    public boolean equals(Object o) {
//          if (!(o instanceof AaslType))
//                return false;
//
//          AaslType obj = (AaslType)o;
//          if (obj.name.compareTo(this.name) != 0)
//                return false;
//
//          return true;
//    }

      public String getType() {
            System.err.println("getType() for generic AaslType not
defined!");
            return "unknown";
      }

      public AaslType copy(){
            return new AaslType();
      }

      public void error(String linenr, String msg){
            throw new AaslException(linenr, "Illegal operation \"" +
msg + "\" : " +
                        getType() + "," + (name!=null ? name: "<null
name>"));
      }

      public void error(String linenr, AaslType b, String msg){
            if(b==null){
                  error(linenr,msg);
                  return;
            }
            throw new AaslException(linenr,"Illegal operation: \"" +
msg + "\" : " +
                        getType() + "," + (name!=null ? name: "<null
name> & ") +
                        b.getType() + "," + (b.name!=null ? b.name:
"<null name>"));
      }
```

```
public void error(String msg){
      this.error("--",msg);
}

public void error(AaslType b, String msg){
      this.error("--",b,msg);
}

public static int intValue(AaslType b){
      if(b instanceof AaslInt)
            return ((AaslInt)b).value;
      if(b instanceof AaslFloat)
            return (int) ((AaslFloat)b).value;
      b.error("cast to int");
      return 0;
}

public static float floatValue(AaslType b){
      if(b instanceof AaslInt)
            return (float)((AaslInt)b).value;
      if(b instanceof AaslFloat)
            return (float) ((AaslFloat)b).value;
      b.error("cast to float");
      return 0;
}

public static boolean checkFloatRange(AaslType b)
{
      if(b instanceof AaslFloat)
      {
            float temp = ((AaslFloat) b).value ;
            if((temp >= 0.0 && temp <= 1.0))
                  return true;
            else
                  return false;
      }
      else if(b instanceof AaslInt && (((AaslInt)b).value == 1))
            return true;
      return false;
}

public AaslType plus(AaslType b, AaslSegArray s, Vector v){
      error(b, "+");
      return null;
}

public AaslType minus(AaslType b, AaslSegArray s){
      error(b, "-");
      return null;
}

public AaslType times(AaslType b, AaslSegArray s){
      error(b, "*");
      return null;
}

public AaslType div(AaslType b, AaslSegArray s){
			error(b, "/");
			return null;
		}

		public AaslType mod(AaslType b){
			error(b, "%");
			return null;
		}

		public AaslType uminus(){
			error("-");
			return null;
		}

		public AaslType assign(AaslType b){
			error(b, "=");
			return null;
		}

		public AaslType not(){
			error("!");
			return null;
		}

		public AaslType lt(AaslType b){
			error(b, "<");
			return null;
		}

		public AaslType gt(AaslType b){
			error(b, ">");
			return null;
		}

		public AaslType le(AaslType b){
			error(b, "<=");
			return null;
		}

		public AaslType ge(AaslType b){
			error(b, ">=");
			return null;
		}

		public AaslType eq(AaslType b){
			error(b, "==");
			return null;
		}

		public AaslType ne(AaslType b){
			error(b, "!=");
			return null;
		}

		public AaslType and(AaslType b){
			error(b, "&&");
			return null;
```

```
	}

	public AaslType or(AaslType b){
		error(b, "||");
		return null;
	}

	public AaslType inc(){
		error("++");
		return null;
	}

	public AaslType dec(){
		error("--");
		return null;
	}
}
```

AaslBool.java

```
/*
*	By Rob Katz, Vaishnav Janardhan
*/
package aasl.intermediateTypes;

public class AaslBool extends AaslType
{
	public boolean value;
	public boolean from_or;

	public AaslBool(boolean value){
		this.value=value;
		this.from_or=false;
	}

	public AaslBool(boolean value, boolean from_or){
		this.value=value;
		this.from_or=from_or;
	}

	public String getType(){
		return "bool";
	}

	public AaslType copy(){
		return new AaslBool(value);
	}

	/*public static boolean boolValue(AaslType b){
		if(b instanceof AaslBool)
			return ((AaslBool)b).value;
		if(b instanceof AaslBool)
			return (boolean) ((AaslBool)b).value;
		b.error("cast to boolean");
		return false;
	}*/

	public AaslType and(AaslType b)
	{
		if (b instanceof AaslBool)
			return new AaslBool(this.value && ((AaslBool)
b).value);
		error(b, "&&");
		return null;
		//b.error("bool && " + b.displayType() + " not supported");
	}

	public AaslType or(AaslType b)
	{
		if (b instanceof AaslBool)
			return new AaslBool(this.value || ((AaslBool)
b).value);
		error(b, "||");
		return null;
		//b.error("bool || " + b.displayType() + " not supported");
	}

	public AaslType not(AaslType b)
	{
		return new AaslBool(!this.value);
	}

	public AaslType eq(AaslType b)
	{
		if (b instanceof AaslBool)
		return new AaslBool((this.value && ((AaslBool) b).value)
				|| (!this.value && !((AaslBool) b).value));
		//b.error("bool == " + b.displayType() + " not supported");
		error(b, "==");
		return null;
	}

	public AaslType ne(AaslType b)
	{
		if (b instanceof AaslBool)
			return new AaslBool((this.value && !((AaslBool)
b).value)

			|| (!this.value && ((AaslBool) b).value));
		error(b, "!=");
		return null;
		//b.error("bool != " + b.displayType() + " not supported");
	}
}
```

AaslInt.java

```
/*
*	By Rob Katz, Vaishnav Janardhan
*/
```

```
package aasl.intermediateTypes;
import aasl.*;
import java.util.*;

public class AaslInt extends AaslType{
	public int value;

	public AaslInt(String value) {
		this.value = Integer.valueOf(value).intValue();
	}
	public AaslInt(int value){
		this.value=value;
	}

	public String getType(){
		return "int";
	}

	public AaslType copy(){
		return new AaslInt(value);
	}

	public int getValue(){
		return value;
	}

	public void setValue(AaslType a){
		value = intValue(a);
	}

	public AaslType plus(AaslType b, AaslSegArray s, Vector v){
		if(b instanceof AaslInt)
			return new AaslInt(value + intValue(b));
		if(b instanceof AaslFloat)
			return new AaslFloat(value + floatValue(b));
		error(b, "illegal addition");
		return null;
	}

	public AaslType minus(AaslType b, AaslSegArray s){
		if(b instanceof AaslInt)
			return new AaslInt(value - intValue(b));
		if(b instanceof AaslFloat)
			return new AaslFloat(value - floatValue(b));
		error(b, "illegal subtraction");
```

```
        return null;
    }

    public AaslType times(AaslType b, AaslSegArray s){
        if(b instanceof AaslInt)
            return new AaslInt(value * intValue(b));
        if(b instanceof AaslFloat)
            return new AaslFloat(value * AaslFloat.floatValue(b));
        if(b instanceof AaslOscillator2){
            return ((AaslOscillator2)b).times(this, s);
        }
        if(b instanceof AaslMixer2){
            return ((AaslMixer2)b).times(this, s);
        }
        error(b, "illegal multiplication");
        return null;
    }

    public AaslType div(AaslType b, AaslSegArray s){
        if(b instanceof AaslInt)
            return new AaslInt(value / intValue(b));
        if(b instanceof AaslFloat)
            return new AaslFloat(value / floatValue(b));
        error(b, "illegal division");
        return null;
    }

    public AaslType mod(AaslType b){
        if(b instanceof AaslInt)
            return new AaslInt(value % intValue(b));
        if(b instanceof AaslFloat)
            return new AaslFloat(value % floatValue(b));
        error(b, "illegal modulus");
        return null;
    }

    public AaslType uminus(){
        return new AaslInt(-value);
    }

    public AaslType lt(AaslType b){
        if(b instanceof AaslInt)
            return new AaslBool(value < intValue(b));
        if(b instanceof AaslFloat)
            return new AaslBool(value < floatValue(b));
        error(b, "illegal comparison");
```

```
		return null;
	}

	public AaslType gt(AaslType b){
		if(b instanceof AaslInt)
			return new AaslBool(value > intValue(b));
		if(b instanceof AaslFloat)
			return new AaslBool(value > floatValue(b));
		error(b, "illegal comparison");
		return null;
	}

	public AaslType le(AaslType b){
		if(b instanceof AaslInt)
			return new AaslBool(value <= intValue(b));
		if(b instanceof AaslFloat)
			return new AaslBool(value <= floatValue(b));
		error(b, "illegal comparison");
		return null;
	}

	public AaslType ge(AaslType b){
		if(b instanceof AaslInt)
			return new AaslBool(value >= intValue(b));
		if(b instanceof AaslFloat)
			return new AaslBool(value >= floatValue(b));
		error(b, "illegal comparison");
		return null;
	}

	public AaslType eq(AaslType b){
		if(b instanceof AaslInt)
			return new AaslBool(value == intValue(b));
		if(b instanceof AaslFloat)
			return new AaslBool(value == floatValue(b));
		error(b, "illegal comparison");
		return null;
	}

	public AaslType ne(AaslType b){
		if(b instanceof AaslInt)
			return new AaslBool(value != intValue(b));
		if(b instanceof AaslFloat)
			return new AaslBool(value != floatValue(b));
		error(b, "illegal comparison");
		return null;
	}

	public AaslType inc(){
		return new AaslInt(value+1);
	}

	public AaslType dec(){
		return new AaslInt(value-1);
	}
}
```

AaslFloat.java

```
/*
*	By Rob Katz, Vaishnav Janardhan
*/

package aasl.intermediateTypes;

import aasl.*;
import java.util.*;

public class AaslFloat extends AaslType
{
	public float value;

	public AaslFloat(String value) {
		this.value = Float.valueOf(value).floatValue();
	}
	public AaslFloat(float value){
		this.value=value;
	}

	public String getType(){
		return "float";
	}

	public AaslType copy(){
		return new AaslFloat(value);
	}

	public AaslType plus(AaslType b, AaslSegArray s, Vector v){
		if(b instanceof AaslFloat)
			return new AaslFloat(value + floatValue(b));
		if(b instanceof AaslInt)
```

```
                return new AaslFloat(value + floatValue(b));
        error(b, "illegal addition");
        return null;
}

public AaslType minus(AaslType b, AaslSegArray s){
        if(b instanceof AaslFloat)
                return new AaslFloat(value - floatValue(b));
        if(b instanceof AaslInt)
                return new AaslFloat(value - floatValue(b));
        error(b, "illegal subtraction");
        return null;
}

public AaslType times(AaslType b, AaslSegArray s){
        if(b instanceof AaslFloat)
                return new AaslFloat(value * floatValue(b));
        if(b instanceof AaslInt)
                return new AaslFloat(value * floatValue(b));
        if(b instanceof AaslOscillator2)
                return b.times(this, s);
        if(b instanceof AaslMixer2)
                return b.times(this, s);
        error(b, "illegal multiplication");
        return null;
}

public AaslType div(AaslType b, AaslSegArray s){
        if(b instanceof AaslInt)
                return new AaslFloat(value / floatValue(b));
        if(b instanceof AaslInt)
                return new AaslFloat(value / floatValue(b));
        error(b, "illegal division");
        return null;
}

public AaslType mod(AaslType b){
        if(b instanceof AaslFloat)
                return new AaslFloat(value % floatValue(b));
        if(b instanceof AaslInt)
                return new AaslFloat(value % floatValue(b));
        error(b, "illegal modulus");
        return null;
}

public AaslType uminus(){
        return new AaslFloat(-value);
}

public AaslType lt(AaslType b){
        if(b instanceof AaslFloat)
                return new AaslBool(value < floatValue(b));
        if(b instanceof AaslInt)
                return new AaslBool(value < floatValue(b));
        error(b, "illegal comparison");
        return null;
}

public AaslType gt(AaslType b){
        if(b instanceof AaslFloat)
                return new AaslBool(value > floatValue(b));
        if(b instanceof AaslInt)
                return new AaslBool(value > floatValue(b));
        error(b, "illegal comparison");
        return null;
}

public AaslType le(AaslType b){
        if(b instanceof AaslFloat)
                return new AaslBool(value <= floatValue(b));
        if(b instanceof AaslInt)
                return new AaslBool(value <= floatValue(b));
        error(b, "illegal comparison");
        return null;
}

public AaslType ge(AaslType b){
        if(b instanceof AaslFloat)
                return new AaslBool(value >= floatValue(b));
        if(b instanceof AaslInt)
                return new AaslBool(value >= floatValue(b));
        error(b, "illegal comparison");
        return null;
}

public AaslType eq(AaslType b){
        if(b instanceof AaslFloat)
                return new AaslBool(value == floatValue(b));
        if(b instanceof AaslInt)
                return new AaslBool(value == floatValue(b));
        error(b, "illegal comparison");
        return null;
```

```
	}

	public AaslType ne(AaslType b){
		if(b instanceof AaslFloat)
			return new AaslBool(value != floatValue(b));
		if(b instanceof AaslInt)
			return new AaslBool(value != floatValue(b));
		error(b, "illegal comparison");
		return null;
	}

	public AaslType inc(){
		return new AaslFloat(value+1);
	}

	public AaslType dec(){
		return new AaslFloat(value-1);
	}
}
```

AaslString.java

```
/*
*	By Rob Katz, Vaishnav Janardhan
*/

package aasl.intermediateTypes;

public class AaslString extends AaslType {
	public String value;

	public AaslString(String value){
		this.value = value;
	}

	public String getType() {
		return "string";
	}
}
```

AaslVariable.java

```
/*
* By Carlos R. Perez,
 */
package aasl.intermediateTypes;

public class AaslString extends AaslType {
	public String value;

	public AaslString(String value){
		this.value = value;
	}

	public String getType() {
		return "string";
	}
}
```

AaslVoid.java

```
/*
 * By Carlos R. Perez,
 */
package aasl.intermediateTypes;


public class AaslVoid extends AaslType
{

	public AaslVoid() {
		this.name = "void";
	}

	public String getType(){
		return "void";
	}
}
```

AaslEnvelope.java

```
/*
*	By Rob Katz, Carlos R. Perez,
*/

package aasl.intermediateTypes;

import java.util.*;

import antlr.collections.AST;


public class AaslEnvelope extends AaslType {
	public Vector v;

	public AaslEnvelope(String name){
		v = new Vector();
		this.name = name;
	}

	/**
	 * Instantiate a AaslEnvelope from the AST sub-tree, all tuples are float!
	 *
	 * @param name
	 * @param floatTuples AST Type
	 */
	public AaslEnvelope(String name, AST floatTuples) {
		this.name = name;
		v = new Vector();

		/* All tuples are float! */
		for (AST floatType=floatTuples; floatType!=null;
floatType=floatType.getNextSibling()) {
			AaslFloat val1 = new AaslFloat(floatType.getText());
			floatType=floatType.getNextSibling();
			AaslFloat val2 = new AaslFloat(floatType.getText());

			this.addElement(val1,val2);
		}
	}

	public String getType(){
		return "envelope";
	}

	public void addElement(AaslFloat time, AaslFloat value){
		AaslEnvelopePair p = new AaslEnvelopePair(floatValue(time),
floatValue(value));
//		System.out.println("["+floatValue(time)+","+floatValue(value)+"]");
		v.add(p);
	}
}
```

AaslEnvelopePair.java

```
/*
*	By Rob Katz,
*/

package aasl.intermediateTypes;

public class AaslEnvelopePair{
	public float time;
	public float value;

	public AaslEnvelopePair(float time, float value){
		if(time>1 || time<0 || value>1 || value <0) this.time=-1;
		if(time>1){
			throw new AaslException("Illegal envelope time 
value:" + time +
						"	Must be less than 1.");
		}
		if(time < 0){
			throw new AaslException("Illegal envelope time 
value:" + time +
					"	Must be greater than 0.");
		}
		if(value>1){
			throw new AaslException("Illegal envelope value:" + 
value +
						"	Must be less than 1.");
		}
		if(value < 0){
			throw new AaslException("Illegal envelope value:" + 
time +
					"	Must be greater than 0.");
		}
		this.time = time;
		this.value = value;
	}
}
```

AaslMixer2.java

```
/*
*	By Rob Katz, Carlos R. Perez,
*/

package aasl.intermediateTypes;

import aasl.*;
import java.util.*;

public class AaslMixer2 extends AaslType{
	public static int temp = 0;
	public float[] factorSum;
	public float[] factor;
	public Vector v;
	public boolean output[];
	public boolean everOutput = false;

	public String getType() {
		return "mixer";
	}

	public AaslMixer2(String name, int segments){
		if(name==null){
			temp++;
			this.name = "temp" + temp;
		}
		else
			this.name = name;
		factorSum = new float[segments];
		factor = new float[segments];
		output = new boolean[segments];
		v = new Vector();
		for(int i=0; i<segments; i++){
			factorSum[i] = 0;
			factor[i] = 1;
			output[i] = false;
		}
	}

	public AaslMixer2(String name, AaslType a, AaslType b, int segments){
		if(name==null){
			temp++;
			this.name = "temp" + temp;
		}
		this.name = name;
		factorSum = new float[segments];
		factor = new float[segments];
		output = new boolean[segments];
		for(int i=0; i<segments; i++)
			output[i] = false;

		if(a instanceof AaslOscillator2 && b instanceof AaslOscillator2){
			for(int i=0; i<segments; i++){
				factorSum[i] = ((AaslOscillator2) a).factor[i] +
							((AaslOscillator2)
b).factor[i];
				v = new Vector();
				v.add(a);
				v.add(b);
			}
		}
		else if(a instanceof AaslOscillator2 && b instanceof AaslMixer2){
			for(int i=0; i<segments; i++){
				factorSum[i] = ((AaslOscillator2) a).factor[i] +
                                                        ((AaslMixer2) b).factor[i];
                        v = new Vector();
                        v.add(a);
                        v.add(b);
                }
        }
        else if(a instanceof AaslOscillator2 && b instanceof
AaslExecutedFunction){
                for(int i=0; i<segments; i++){
                        factorSum[i] = ((AaslOscillator2) a).factor[i] +
                                                        ((AaslExecutedFunction)
b).factor[i];
                        v = new Vector();
                        v.add(a);
                        v.add(b);
                }
        }
        else if(a instanceof AaslMixer2 && b instanceof AaslExecutedFunction){
                for(int i=0; i<segments; i++){
                        factorSum[i] = ((AaslMixer2) a).factor[i] +
                                                        ((AaslExecutedFunction)
b).factor[i];
                        v = new Vector();
                        v.add(a);
                        v.add(b);
                }
        }
        else if(b instanceof AaslOscillator2 && a instanceof
AaslExecutedFunction){
                for(int i=0; i<segments; i++){
                        factorSum[i] = ((AaslOscillator2) b).factor[i] +
                                                        ((AaslExecutedFunction)
a).factor[i];
                        v = new Vector();
                        v.add(a);
                        v.add(b);
                }
        }
        else if(b instanceof AaslMixer2 && a instanceof AaslExecutedFunction){
                for(int i=0; i<segments; i++){
                        factorSum[i] = ((AaslMixer2) b).factor[i] +
                                                        ((AaslExecutedFunction)
a).factor[i];
                        v = new Vector();
                        v.add(a);
                        v.add(b);
```

```
				}
			}
			else if(a instanceof AaslExecutedFunction && b instanceof
AaslExecutedFunction){
				for(int i=0; i<segments; i++){
					factorSum[i] = ((AaslExecutedFunction)a).factor[i] +

		((AaslExecutedFunction)b).factor[i];
					v = new Vector();
					v.add(a);
					v.add(b);
				}
			}
			else if(a instanceof AaslMixer2 && b instanceof AaslOscillator2){
				for(int i=0; i<segments; i++){
					factorSum[i] = ((AaslMixer2) a).factor[i] +
									((AaslOscillator2)
b).factor[i];
					v = new Vector();
					v.add(a);
					v.add(b);
				}
			}
			else if(a instanceof AaslMixer2 && b instanceof AaslMixer2){
				for(int i=0; i<segments; i++){
					factorSum[i] = ((AaslMixer2) a).factor[i] +
									((AaslMixer2) b).factor[i];
					v = new Vector();
					v.add(a);
					v.add(b);
				}
			}
			else{
				error("<Bad Constructor Arguments>");
			}

			for(int i=0; i<segments; i++){
				factor[i] = 1;
			}
		}

		public AaslType copy(){
			AaslMixer2 mix = new AaslMixer2(this.name, this.factor.length);
			mix.factorSum = this.factorSum;
			mix.factor = this.factor;
			mix.v = this.v;
		mix.output = this.output;
		mix.everOutput = false;
		return mix;
	}

	public int numElements(){
		return v.size();
	}

	public void setAsOutput(AaslSegArray s){
		for(int i=0; i<output.length; i++){
			if(s.getValue(i)){
				output[i]=true;
				everOutput = true;
			}
		}
	}

	public void addElement(AaslType b){
		if(b instanceof AaslOscillator2){
			for(int i=0; i<factorSum.length; i++){
				factorSum[i] += ((AaslOscillator2)b).factor[i];
			}
			v.add(b);
		}
		else if(b instanceof AaslMixer2){
			for(int i=0; i<factorSum.length; i++){
				factorSum[i] += ((AaslMixer2)b).factor[i];
			}
			v.add(b);
		}
		else if(b instanceof AaslExecutedFunction){
			for(int i=0; i<factorSum.length; i++){
				factorSum[i] += ((AaslExecutedFunction)b).factor[i];
			}
			v.add(b);
		}
		else{
			error(b, "<Add to mixer>");
		}
	}

	public AaslType plus(AaslType b, AaslSegArray s,Vector v){
		if(b instanceof AaslOscillator2)
			return ((AaslOscillator2)b).plus(this,s,v);
		else if(b instanceof AaslMixer2){
				this.addElement(b);
				return this;
			}
			else{
				error(b, "+");
				return null;
			}
		}
}
```

AaslOscillator2.java

```
/*
*	By Rob Katz, Carlos R. Perez, Vaishnav Janardhan
*/

package aasl.intermediateTypes;

import java.util.*;
import aasl.*;

public class AaslOscillator2 extends AaslType{
	public String waveType[];
	public AaslType freq[];
	public AaslType amp[];
	public float centralFreq[];
	public float factor[]; //for use in Mixer assignment

	public AaslOscillator2(String name, String waveType, int segments,
						AaslSegArray s){
		checkString(waveType);
		this.name = name;
		this.waveType = new String[segments];
		freq = new AaslType[segments];
		amp = new AaslType[segments];
		centralFreq = new float[segments];
		factor = new float[segments];
		for(int i=0; i<segments; i++){
			if(s.getValue(i))
				this.waveType[i] = waveType;
			else
				this.waveType[i] = null;
			freq[i] = null;
			amp[i] = null;
			centralFreq[i] = 0;
```

```
                factor[i]=1;
        }
}

public AaslOscillator2(String name, String waveType, AaslType freq,
                AaslType amp, float centralFreq, int segments){
        checkString(waveType);
        this.name = name;
        this.waveType = new String[segments];
        this.freq = new AaslType[segments];
        this.amp = new AaslType[segments];
        this.factor = new float[segments];
        this.centralFreq = new float[segments];
        for(int i=0; i<segments; i++){
                this.waveType[i] = waveType;
                this.freq[i] = freq;
                this.amp[i] = amp;
                this.factor[i] = 1;
                this.centralFreq[i] = centralFreq;
        }
}

public AaslOscillator2(String name, String[] waveType, AaslType[] freq,
                AaslType[] amp, float centralFreq[], int segments){
        checkString(waveType);
        this.name = name;
        this.waveType = new String[segments];
        this.freq = new AaslType[segments];
        this.amp = new AaslType[segments];
        this.centralFreq = new float[segments];
        this.factor = new float[segments];
        for(int i=0; i<segments; i++){
                this.waveType[i] = waveType[i];
                this.freq[i] = freq[i];
                this.amp[i] = amp[i];
                this.centralFreq[i] = centralFreq[i];
                this.factor[i] = 1;
        }
}

public AaslOscillator2(String name, String waveType, AaslType freq,
                AaslType amp, float centralFreq, float factor, int segments){
        checkString(waveType);
        this.name = name;
        this.waveType = new String[segments];
        this.freq = new AaslType[segments];
```

```
		this.amp = new AaslType[segments];
		this.centralFreq = new float[segments];
		this.factor = new float[segments];
		for(int i=0; i<segments; i++){
			this.waveType[i] = waveType;
			this.freq[i] = freq;
			this.amp[i] = amp;
			this.centralFreq[i] = centralFreq;
			this.factor[i] = factor;
		}
	}

	public AaslOscillator2(String name, String[] waveType, AaslType[] freq,
			AaslType[] amp, float[] centralFreq, float[] factor, int segments){
		checkString(waveType);
		this.name = name;
		this.waveType = new String[segments];
		this.freq = new AaslType[segments];
		this.amp = new AaslType[segments];
		this.centralFreq = new float[segments];
		this.factor = new float[segments];
		for(int i=0; i<segments; i++){
			this.waveType[i] = waveType[i];
			this.freq[i] = freq[i];
			this.amp[i] = amp[i];
			this.centralFreq[i] = centralFreq[i];
			this.factor[i] = factor[i];
		}
	}

	public AaslType copy(){
		return new AaslOscillator2(name, waveType, freq, amp, centralFreq,
				factor, factor.length);
	}

	public String getType(){
		return "oscillator";
	}

	/*public AaslType copy(){
		return new AaslOscillator2(waveType, freq, amp, factor, factor.length);
	}*/

	public void setFactor(float factor){
		for(int i=0; i<this.factor.length; i++){
			this.factor[i] = factor;
			}
		}

		public void setFactor(float factor, int segment){
			this.factor[segment] = factor;
		}

		public void setWaveType(AaslString string, AaslSegArray s){
			checkString(string.value);
			for(int i=0; i<waveType.length; i++){
				if(s.getValue(i))
					waveType[i]=string.value;
			}
		}

		public AaslType plus(AaslType b, AaslSegArray s, Vector v){
			if(b instanceof AaslOscillator2){
				AaslOscillator2 a = (AaslOscillator2)this.copy();
				AaslOscillator2 bcopy = (AaslOscillator2)b.copy();
				for(int i=0; i<s.length; i++){
					if(!s.getValue(i)){
						a.factor[i] = 0;
						bcopy.factor[i] = 0;
					}
				}
				AaslMixer2 mix = new AaslMixer2(null, a, b, a.factor.length);
				return mix;
			}
			else if(b instanceof AaslMixer2){
				v.remove(this);
				AaslOscillator2 a = (AaslOscillator2)this.copy();
				for(int i=0; i<s.length; i++){
					if(!s.getValue(i))
						a.factor[i]=0;
				}
				((AaslMixer2)b).addElement(a);
				v.add(b);
				return b;
			}
			else if(b instanceof AaslExecutedFunction) {
				AaslOscillator2 a = (AaslOscillator2)this.copy();
				AaslExecutedFunction bcopy =
((AaslExecutedFunction)b).clone();

				for(int i=0; i<s.length; i++){
					if(!s.getValue(i)){
```

```
                    a.factor[i] = 0;
                    bcopy.factor[i] = 0;
                }
            }
            AaslMixer2 mix = new AaslMixer2(null, a, b, a.factor.length);
            return mix;
        } else {
            error(b, "Cannot add AaslOscillator2 with <"+b.getType()+">");
            return null;
        }
    }
    ///////////Can you multipy two oscillators to form a mixer
    public AaslType times(AaslType b, AaslSegArray s){
        float[] new_factor = new float[factor.length];
        if(b instanceof AaslInt){
            for(int i=0; i<factor.length; i++){
                if(s.getValue(i))
                    new_factor[i] = factor[i] * ((AaslInt)b).value;
                else
                    new_factor[i] = factor[i];
            }
            return new AaslOscillator2(name, waveType, freq, amp,
centralFreq,
                    new_factor, factor.length);
        }
        else if(b instanceof AaslFloat){
            for(int i=0; i<factor.length; i++){
                if(s.getValue(i))
                    new_factor[i] = factor[i] * ((AaslFloat)b).value;
                else
                    new_factor[i] = factor[i];
            }
            return new AaslOscillator2(name, waveType, freq, amp,
centralFreq,
                    new_factor,     factor.length);
        }
        else{
            error(b, "*");
            return null;
        }
    }

    private void checkString(String waveType){
        if( waveType.compareTo("SINE")!=0 &&
            waveType.compareTo("SQUARE")!=0 &&
            waveType.compareTo("SAW")!=0 &&
                        waveType.compareTo("REVSAW")!=0)
                        throw new AaslException(waveType + " is not a valid
wavetype.");
        }

        private void checkString(String[] waveTypeArr){
                for(int i=0; i<waveTypeArr.length; i++)
                {
                        String waveType = waveTypeArr[i];
                        if( waveType.compareTo("SINE")!=0 &&
                                        waveType.compareTo("SQUARE")!=0 &&
                                        waveType.compareTo("SAW")!=0 &&
                                        waveType.compareTo("REVSAW")!=0)
                                        throw new AaslException(waveType + " is not a
valid wavetype.");
                }

        }
}
```

AaslOscillatorBank.java

```
/*
*       By Rob Katz, Vaishnav Janardhan
*/

package aasl.intermediateTypes;

import aasl.AaslSegArray;

public class AaslOscillatorBank extends AaslType
{
      AaslOscillator2[] oscbnk_array = null;

      public AaslOscillatorBank()
      {
      }

      public AaslOscillatorBank(int number,String name, String
waveType, int segments,AaslSegArray s)
      {
            oscbnk_array = new AaslOscillator2[number];

            for(int i=0; i < number; i++)
            {
                  this.oscbnk_array[i] = new AaslOscillator2(name+((new
Integer(i)).toString()),
                                    waveType,segments,s);
            }
      }
```

```
	public AaslOscillator2[] getArray()
	{
		return this.oscbnk_array;
	}
	public String convertBankToOSCElem(String osc_block,String index)
	{
		return (osc_block+((new Integer(index)).toString()));
	}

	public String getType() {
		return "oscillator bank";
	}
}
```

SynthElement.java

```
/*
*	By Rob Katz
*/

package synthEngine;



public class SynthElement {
	public static final int SAMPRATE = 44100;

	static double[][] WAVEFORMS = new double[4][SAMPRATE];
	/*static double[] SINEWAVE = new double[SAMPRATE];
	static double[] SQUAREWAVE = new double[SAMPRATE];
	static double[] SAWWAVE = new double[SAMPRATE];
	static double[] REVSAWWAVE = new double[SAMPRATE];*/

	static final int SINE = 0;
	static final int SQUARE = 1;
	static final int SAW = 2;
	static final int REVSAW = 3;

	public SynthElement(){
		for(int i=0; i<SAMPRATE; i++)
			{
				//initialize sinewave array
				WAVEFORMS[0][i] = (double)
Math.sin(2*Math.PI*i/SAMPRATE);

				//initialize squarewave array
				if( i < (SAMPRATE/2) )
					WAVEFORMS[SQUARE][i]=1;
				else
					WAVEFORMS[SQUARE][i]=-1;
				double j = (double)i;
				//initialize SAW and REVSAW arrays
				WAVEFORMS[SAW][i] = 1 - 2*(j/SAMPRATE);
				WAVEFORMS[REVSAW][i] = -1 * WAVEFORMS[SAW][i];
```

```
                  }
      }//SynthElement()
}
```

Oscillator.java

```
/*
*       By Rob Katz,
*/

package synthEngine;



public class Oscillator extends SynthElement {



      double c_freq;          //central frequency
      String waveString;      //waveString type
      int wavetype = 0;
      double[] freq_input; //array of values used to modify frequency
      double[] amp_input;     //array of values used to modify
amplitude
      public double[] output;       //array to hold output waveString
      double const_freq;
      double const_amp;
      boolean is_freq_const = false;
      boolean is_amp_const = false;
      boolean amp_is_env;
      double waveString_index = 0;

      double currentPlace = 0;
      double currentFreq = c_freq;

      public Oscillator(double cfreq, String wave, double tone_length,
                  double[] freqArr, double[] ampArr, boolean
amp_is_env)
      {
            c_freq = cfreq;
            setWavetype(wave);
            output = new double[(int)(SAMPRATE * tone_length)];
            freq_input = freqArr;
            amp_input = ampArr;
            const_freq = 0;
            const_amp = 0;
            this.amp_is_env = amp_is_env;
            this.generate_output();
      }

      public Oscillator(double cfreq, String wave, double tone_length,
                  double[] freqArr, double amp)
      {
            c_freq = cfreq;
            setWavetype(wave);
            output = new double[(int)(SAMPRATE * tone_length)];
            freq_input = freqArr;
            const_freq = 0;
            const_amp = amp;
            is_amp_const = true;
            this.generate_output();
      }

      public Oscillator(String wave, double tone_length,
                  double freq, double[] ampArr, boolean amp_is_env)
      {
            c_freq = 0;
            setWavetype(wave);
            output = new double[(int)(SAMPRATE * tone_length)];
            const_freq = freq;
            amp_input = ampArr;
            const_amp = 0;
            is_freq_const = true;
            this.amp_is_env = amp_is_env;
            this.generate_output();
      }

      public Oscillator(String wave, double tone_length,
                  double freq, double amp)
      {
            c_freq = 0;
            setWavetype(wave);
            output = new double[(int)(SAMPRATE * tone_length)];
            const_freq=freq;
            const_amp = amp;
            is_freq_const = true;
            is_amp_const = true;
            this.generate_output();
      }

      public void generate_output()
      {
            double sample;
            for(int i=0; i<output.length; i++)
            {
                  sample = WAVEFORMS[wavetype][(int)currentPlace];

                  if(!is_amp_const && (amp_input.length > i))
                  {
                        if(amp_is_env)
                        {
                              sample *= amp_input[i];
                        }
                        else
                        {
                              sample = sample * 0.5 * (amp_input[i] +
1);
                        }
                  }
                  else
                        sample *= const_amp;
```

```
			if(!is_freq_const && (freq_input.length > i) )
				currentFreq = (c_freq/2) * Math.pow(2, (1 + 
freq_input[i]));
			else
				currentFreq = const_freq;

			currentPlace = (currentPlace + (currentFreq * 
(SAMPRATE/44100)) ) % SAMPRATE;
			if(sample>1.0)
				sample=0;
			output[i] = sample;
		}
	}

	private void setWavetype(String waveString)
	{
		if(waveString.equals("SINE"))
		{
			wavetype = SINE;
		}
		else if(waveString.equals("SQUARE"))
		{
			wavetype = SQUARE;
		}
		else if(waveString.equals("SAW"))
		{
			wavetype = SAW;
		}
		else if(waveString.equals("REVSAW"))
		{
			wavetype = REVSAW;
		}
	}
}
```

Envelope.java

```
/*
*	By Rob Katz,
*/

package synthEngine;



public class Envelope extends SynthElement{

	public double[] output;
	int i, j;
	double tot_samples, factor;
	//input pairs (time,value)
	public Envelope(double[][] input, int num_points, double 
tone_length)
	{
```

```
            tot_samples = tone_length * SAMPRATE;
            output = new double[(int)(tot_samples)];
            input_sort(input, num_points);

            j=0;
            i=0;
            if(input[0][0] != 0.0)
            {
                  while(j < (input[0][0]*tot_samples)  )
                  {
                        output[j] = input[0][1] *
(j/(input[0][0]*tot_samples));
                        ++j;
                  }
                  i=1;
            }
            for(; i<num_points-1; i++)
            {
                  while(j < ((input[i+1][0]) * tot_samples))
                  {
                        factor = ( (j - (input[i][0] *tot_samples)) /
                                    ( (input[i+1][0]-input[i][0]) *
tot_samples) );
                        output[j] = input[i][1] + ( (input[i+1][1] -
input[i][1]) * factor);
                        ++j;
                  }
            }
            if(input[num_points-1][0] != 1.0)
            {
                  while(j<tot_samples)
                  {
                        output[j] = input[num_points-1][1];
                        j++;
                  }
            }
      }

      public double[] output(int seg, int segments){
            double[] ret_array = new double[(int)tot_samples/segments];
            for(int i=0; i<ret_array.length; i++){
                  ret_array[i] = output[(seg*ret_array.length)+i];
            }
            return ret_array;
      }

      private void input_sort(double[][] input, int num_points)
      {
            double temp1, temp2;
            for(int i=0; i<num_points; i++)
            {
                  for(int j=i+1; j<num_points; j++)
                  {
                        if(input[i][0] > input[j][0])
                        {
                              temp1 = input[i][0];
                              temp2 = input[i][1];
```

```
                                input[i][0] = input[j][0];
                                input[i][1] = input[j][1];
                                input[j][0] = temp1;
                                input[j][1] = temp2;
                          }
                    }
              }
      }//input_sort end
}
```

BranchInstruction.java

```
/**
   •   Vaishnav Janardhan
   */
package aasl.intermediateTypes;

import aasl.AaslSymbolTable;



public class BranchInstruction extends Instruction {


      public Comparison comp;

      public AaslType ifInst;

      public AaslType elseInst;

      private AaslSymbolTable smbTable;

      public BranchInstruction(Comparison comp, AaslType ifInst,
AaslType elseInst,AaslSymbolTable s) {
            this.comp = comp;
            this.ifInst = ifInst;
            this.elseInst = elseInst;

            if(!usesVariable(ifInst) || !usesVariable(this.elseInst) ||
                        !usesVariable(this.comp.arg1) ||
!usesVariable(this.comp.arg2))
            {
                  throw new Error("Variable not declared");
            }
      }

            public int getInstType()
            {
                  return Instruction.INST_TYPE_BRANCH;
            }

      public boolean usesVariable(AaslType v)
      {
            return smbTable.containsSymbol(v.name);
      }
}
```

Comparision.java

```
package aasl.intermediateTypes;

public class Comparison {

      public final static int CONDITION_GREATER = 0;

      public final static int CONDITION_LESS    = 1;


      public final static int CONDITION_GE      = 2;


      public final static int CONDITION_LE      = 3;


      public final static int CONDITION_EQUAL   = 4;


      public final static int CONDITION_NEQUAL  = 5;


      public AaslType arg1;


      public AaslType arg2;


      public int condition = -1 ;


      public Comparison(AaslType arg1, AaslType arg2, int condition) {
            if (condition!=CONDITION_GREATER &&
condition!=CONDITION_LESS && condition!=CONDITION_GE &&
condition!=CONDITION_LE && condition!=CONDITION_EQUAL &&
condition!=CONDITION_NEQUAL) {
                  throw new Error("invalid condition");
            }
            if (arg1==null) {
                  throw new Error("arg1 is null");
            }
            if (arg2==null) {
                  throw new Error("arg2 is null");
            }



            this.arg1 = arg1;
            this.arg2 = arg2;
            this.condition = condition;
```

```
                AaslType temp = this.arg1;
                this.arg2 = this.arg1;
                this.arg1 = temp;

        }
}
```

ForLoopInstruction.java

```
/*
   •   By Carlos R. Perez,
   */
package aasl.intermediateTypes;

import aasl.AaslAntlrWalker;
import aasl.AaslSegArray;
import aasl.AaslSymbolTable;
import antlr.RecognitionException;
import antlr.collections.AST;

public class ForLoopInstruction {

        private AaslSymbolTable old_symt;
        private AST assignment, bool, delta, for_body;
        private AaslAntlrWalker walker;

        public ForLoopInstruction(AaslAntlrWalker walker, AST assignment, AST bool,
                        AST delta, AST for_body) throws antlr.RecognitionException {
                this.walker = walker;
                this.assignment = assignment;
                this.bool = bool;
                this.delta = delta;
                this.for_body = for_body;

                /*
                 * declare new symbol table block
                 */
                old_symt = walker.symt;
                walker.symt = new AaslSymbolTable( walker.symt );

                /*
                 * initialize assignment expression
                 */
                walker.expr(assignment);
        }
```

```
	public AaslType run() throws RecognitionException {
		/*
		 * check the boolean, do code, repeat until boolean is false.
		 */
		AaslType retValue = new AaslBool(false);

		while( checkCondition() ) {
			retValue = walker.expr(for_body);
			delta();
		}

		restoreSymbolTable();
		return retValue;
	}

	private void restoreSymbolTable() {
		walker.symt = old_symt;
	}

	private void delta() throws RecognitionException {
		walker.expr(delta);
	}

	private boolean checkCondition() throws RecognitionException {
		AaslType boolValue = walker.expr(bool);

		if (boolValue == null)
			throw new AaslException("Evaluation of 'for conditional' as an
expr() returned NULL type!");

		if (boolValue instanceof AaslBool) {
			AaslBool retValue = (AaslBool) boolValue;
			return retValue.value;
		} else
			throw new AaslException("Unsupported type
<"+boolValue.name+"> in 'for conditional'!");
	}
}
```

Instruction.java

```
package aasl.intermediateTypes;

public abstract class Instruction {
	public final static int INST_TYPE_BRANCH = 0;


	public final static int INST_TYPE_FORLOOP = 1;


	public abstract int getInstType();


	/** To check if the variable is defined within the scope */
	public abstract boolean usesVariable(AaslType v);


}
```

LoopInstruction.java

```
package aasl.intermediateTypes;

import aasl.AaslSymbolTable;

public class LoopInstruction extends Instruction
{


	public AaslType assignmentInstr;

	public AaslType loopInstr;
	private AaslSymbolTable smbTable = null;

	public LoopInstruction(AaslType assignmentInstr, AaslType
loopInstr,AaslSymbolTable s) {
		this.assignmentInstr = assignmentInstr;

		this.loopInstr = loopInstr;
		this.smbTable = s;

		if( !usesVariable(this.assignmentInstr) ||
!usesVariable(this.loopInstr))

			throw new Error("Variable Not found");
	}


	public boolean usesVariable(AaslType v)
	{
		return this.smbTable.containsSymbol(v.name);

	}

	public int getInstType() {
		return Instruction.INST_TYPE_FORLOOP;
	}
}
```

Segment.java

```
package aasl.intermediateTypes;

public class Segment
{
	Segment()
	{
	
	}
	
	int[] getSegSet(String segset)
	{
	
		return null;
	}
	

}
```

AaslException.java

```

/*
 * AaslException.java
 * By Rob Katz, Carlos R. Perez,
 *
 */

package aasl.intermediateTypes;

public class AaslException extends RuntimeException{


	/**
	 * Set to 'false' on production/demo USE
	 * Set to 'true' on compiler debugging USE
	 */
	public static boolean DEBUG = true;
	public AaslException(String msg) {
		System.err.println("<-AaslException thrown->" + msg);
		if(!DEBUG)
			System.exit(-1);
	}
	public AaslException(String linenr, String msg) {
		System.err.println("["+linenr+"] "+msg);
		if(!DEBUG)
			System.exit(-1);
	}
}
```

AaslExecutedFunction.java

```
/*
 * AaslType.java
 *
 * By Carlos R. Perez, Rob Katz.
 *
 */

package aasl.intermediateTypes;

import java.util.Vector;

import aasl.AaslSegArray;


public class AaslExecutedFunction extends AaslType{

	String functionName;
	AaslFunctionCallArguments fcallArgs;
	AaslFunctionDef thisFuncDef;
	Vector v;
	int seg;
	public float[] factor;

	public String getType() {
		return "executed function call";
	}

	private AaslExecutedFunction() {
		functionName = null;
		fcallArgs = null;
		thisFuncDef = null;
		v = null;
		factor = null;
	}

	public AaslExecutedFunction(String fName, AaslFunctionCallArguments fargs,
AaslFunctionDef fdef, Vector fV, int seg) {
		functionName = fName;
		name = fName + this.hashCode();
		fcallArgs = fargs;
		thisFuncDef = fdef;
		v = fV;
		this.seg = seg;
		factor = new float[seg];
		for(int i=0; i<seg; i++)
			factor[i]=1;
	}

	public Vector getV(){
		return this.v;
	}

	/**
	 * Creates a Mixer with both types embedded
	 */
	public AaslType plus(AaslType b, AaslSegArray s, Vector v){

		if(b instanceof AaslOscillator2) {
			AaslExecutedFunction a = (AaslExecutedFunction)this.clone();
			AaslOscillator2 bcopy = (AaslOscillator2)b.copy();
			for(int i=0; i<s.length; i++){
				if(!s.getValue(i)){
					a.factor[i] = 0;
					bcopy.factor[i] = 0;
				}
			}
			AaslMixer2 mix = new AaslMixer2(null, a, b, a.factor.length);
			return mix;
		} else if (b instanceof AaslMixer2) {
			//v.remove(this);
			AaslExecutedFunction a = (AaslExecutedFunction)this.clone();
			for(int i=0; i<s.length; i++){
				if(!s.getValue(i))
					a.factor[i]=0;
			}
			((AaslMixer2)b).addElement(a);
			//v.add(b);
```

```
            return b;
        } else if (b instanceof AaslExecutedFunction) {
            AaslExecutedFunction a = (AaslExecutedFunction)this.clone();
            AaslExecutedFunction bcopy =
((AaslExecutedFunction)b).clone();
            for(int i=0; i<s.length; i++){
                if(!s.getValue(i)){
                    a.factor[i] = 0;
                    bcopy.factor[i] = 0;
                }
            }
            AaslMixer2 mix = new AaslMixer2(null, a, b, a.factor.length);
            return mix;
        } else {
            error(b, "Cannot add AaslExecutedFunction with
<"+b.getType()+">");
            return null;
        }
    }

    /**
     * Multiply to change AaslExecutedFunction factor
     *
     * INT * FUNCCAL
     * FLOAT * FUNCCAL
     */
    public AaslType times(AaslType b, AaslSegArray s) {

        float[] new_factor = new float[factor.length];
        AaslExecutedFunction newExecFunc = this.clone();

        if(b instanceof AaslInt) {
            for(int i=0; i<factor.length; i++) {
                if(s.getValue(i))
                    new_factor[i] = factor[i] * ((AaslInt)b).value;
                else
                    new_factor[i] = factor[i];
            }
            newExecFunc.factor = new_factor;
        }
        else if(b instanceof AaslFloat) {
            for(int i=0; i<factor.length; i++) {
                if(s.getValue(i))
                    new_factor[i] = factor[i] * ((AaslFloat)b).value;
                else
                    new_factor[i] = factor[i];
				}
				newExecFunc.factor = new_factor;
			}
			else{
				error(b, "Cannot multiply AaslExecutedFunction with
<"+b.getType()+">");
				return null;
			}

			return newExecFunc;
		}

		public AaslExecutedFunction clone() {
			AaslExecutedFunction ret = new AaslExecutedFunction();

			ret.functionName = this.functionName;
			ret.fcallArgs = this.fcallArgs.clone();
			ret.thisFuncDef = this.thisFuncDef.clone();
			ret.v = (Vector)this.v.clone();
			ret.factor = new float[this.seg];
			for (int i=0; i<this.seg; i++)
				ret.factor[i] = this.factor[i];
			ret.name = this.name;
			return ret;
		}
}
```

AaslFunctionCall.java

```
/*
 * Carlos R. Perez,
*/
package aasl.intermediateTypes;

import java.util.Enumeration;
import java.util.Vector;

import aasl.AaslAntlrWalker;
import aasl.AaslSymbolTable;
import antlr.RecognitionException;


/**
 * Wrapper class for a function call
 */
public class AaslFunctionCall extends AaslType
```

```
{
	final String functionName;
	final AaslFunctionCallArguments fcallArgs;
	final AaslSymbolTable curSymbTable;
	final Vector fdefV;
	AaslFunctionDef thisFuncDef;

	/*
	Constructor( <vector> of function definitions,
				*		<SymbolTable> of previous scope used for
argument substitution,
				*		<String> function name,
<AaslFunctionCallArguments>
				*/
	public AaslFunctionCall(Vector fdefV, AaslSymbolTable symbTable, String
funcName, AaslFunctionCallArguments fargs ) {
		this.fdefV = fdefV;
		curSymbTable = symbTable;
		functionName = funcName;
		this.fcallArgs = fargs;

		/* sets 'thisFuncDef' and 'functionName'*/
		checkFuncDefined();
	}

	/*
	 * Support function overloading <polymorphism>
	 */
	private void checkFuncDefined() {
		Vector<AaslFunctionDef> matchingFuncDefs = new
Vector<AaslFunctionDef>();


		/*
		 * THIS would only be useful if they were all variables udef(x) but
		 * not constants
		 *
		 * Check that the function arguments are valid (in the symbol table)!
		 */
		{
		Enumeration<String> fcallArgs_enum = fcallArgs.args.elements();
		while (fcallArgs_enum.hasMoreElements()) {
			String curArgument = fcallArgs_enum.nextElement();
			boolean isVariable = false;

			/* check that the argument is a VARIABLE!!! */
```

```
                        try { Float.valueOf(curArgument); }
                        catch (NumberFormatException e) {
                                isVariable = true;
                        }

                        if (isVariable && !curSymbTable.containsSymbol(curArgument))
                                throw new AaslException("Illegal operation <function call,
argument not defined>: " +
                                                "calling
\""+this.functionName+fcallArgs.toString()+" with argument="+
                                                curArgument+" that has not been defined!");
                }
                }


                /*
                 * Check that function is defined fargs
                 */
                for (int i=0; i < fdefV.size(); i++) {
                        AaslFunctionDef fDef = (AaslFunctionDef)fdefV.elementAt(i);

                        // Add to matchingFuncDefs if the names match.
                        if (fDef.getFunctionName().compareTo(this.functionName) == 0)
                                matchingFuncDefs.add(fDef);
                }
                if (matchingFuncDefs.size() == 0) /* Didn't find a match! */
                        throw new AaslException("Illegal operation <function not
declared>: "
                                        +"calling \""+this.functionName+"\""+"that has not
been declared.");



                /*
                 * Remove function definitions that don't have the same cardinality.
                 */
        {
                Vector<AaslFunctionDef> matchingFuncDefsCOPY =
(Vector<AaslFunctionDef>) matchingFuncDefs.clone();
                Enumeration<AaslFunctionDef> matchingFuncDefs_enum =
matchingFuncDefs.elements();
                while (matchingFuncDefs_enum.hasMoreElements()) {
                        AaslFunctionDef curFuncDef =
matchingFuncDefs_enum.nextElement();

                        if ( curFuncDef.getNumArgs() != fcallArgs.getNumArgs() )
```

```
					matchingFuncDefsCOPY.remove(curFuncDef);
			}
			if (matchingFuncDefs.size() == 0) /* Didn't find a match! */
				throw new AaslException("Illegal operation <function not
declared>: "
						+"calling \""+this.functionName+"\""+"that has not
been declared.");
			matchingFuncDefs = matchingFuncDefsCOPY;
		}



			/*
			 * We have a Vector of function definitions that match by name,
			 * now lets match by function arguments. If the function doesn't
			 * match, we remove it from the vector. Until we end up with one
			 * definition (non-ambigious, can't have >1) or 0 definitions match.
			 *
			 * Compare the function arguments, req. symbolTable
			 */
		{
			/*
			 * Save function call types in a Vector.
			 */
			Vector<AaslType> curFuncCallTypes = new Vector<AaslType>();
			for (int i=0; i < this.fcallArgs.getNumArgs(); i++) {
				String varName = fcallArgs.args.elementAt(i);		//automatically
to string???
				boolean isVariable = false;

				/* determine if its an INT, FLOAT or VARIABLE */
				try {
					Integer.valueOf(varName);
					curFuncCallTypes.add(new AaslInt(varName));
				} catch (NumberFormatException e) {
					try {
						Float.valueOf(varName);
						curFuncCallTypes.add(new AaslFloat(varName));
					} catch (NumberFormatException e2) {
						isVariable = true;
					}
				}

				// Lookup type in Symbol Table
				if (isVariable) {
```

```
				AaslType varType = 
this.curSymbTable.getValue(varName);
				curFuncCallTypes.add(varType);
			}
		}

		Enumeration<AaslFunctionDef> matchingFuncDefs_enum = 
matchingFuncDefs.elements();
		while (matchingFuncDefs_enum.hasMoreElements()) {
			AaslFunctionDef curFuncDef = 
matchingFuncDefs_enum.nextElement();
			AaslFunctionDefArgList curFuncDefPrototype = 
curFuncDef.functionPrototype;

			/*
			 * Save the prototype types in a Vector.
			 */
			Vector<String> curFuncDefTypes = new Vector<String>();
			for (int i=0; i<curFuncDef.getNumArgs(); i++) {
				Vector<String> curFuncDefVarNames = 
curFuncDefPrototype.keys;
				String curType = 
curFuncDefPrototype.getType(curFuncDefVarNames.elementAt(i));

				curFuncDefTypes.add(curType);
			}

			/*
			 * Check the function argument types. If NOT then remove from 
list.
			 */
			Vector<AaslFunctionDef> matchingFuncDefsCOPY = 
(Vector<AaslFunctionDef>) matchingFuncDefs.clone();
			for (int i=0; i<curFuncCallTypes.size(); i++) {
				AaslType curFuncCallType = 
curFuncCallTypes.elementAt(i);
				String curFuncDefType  = curFuncDefTypes.elementAt(i);

				if ( 
curFuncCallType.getType().compareTo(curFuncDefType) != 0 )
					matchingFuncDefsCOPY.remove(curFuncDef);
			}
			matchingFuncDefs = matchingFuncDefsCOPY;
		}

		if (matchingFuncDefs.size() > 1)  {
```

```
                        StringBuffer output = new StringBuffer();
                        output.append("Illegal operation <multiple function declarations
match>:"
                                        +"calling
\""+this.functionName+this.fcallArgs.toString()+"\" matches the"
                                        + " following functions: ");
                        Enumeration<AaslFunctionDef> matchingFuncDefs_enumTMP =
matchingFuncDefs.elements();
                        while (matchingFuncDefs_enumTMP.hasMoreElements()) {
                                AaslFunctionDef curDef =
matchingFuncDefs_enumTMP.nextElement();
                                output.append(curDef.toString());
                                output.append("\n");
                        }

                        throw new AaslException(output.toString());
                }
                if (matchingFuncDefs.size() == 0)/* Didn't find a match! */
                        throw new AaslException("Illegal operation <function not
declared>: "
                                        +"calling
\""+this.functionName+"("+this.fcallArgs.toString()+")\""+" that has not been declared"
+
                                                        "because the declared functions don't
match the requested parameters!");
        }
                //what about if size()>1 ???
                thisFuncDef = matchingFuncDefs.elementAt(0);
                /* success */
        }

        /**
         * Call function!
         * @param walker
         * @param parent - support dynamic scoping
         * @return
         */
        public AaslExecutedFunction evaluate(AaslAntlrWalker walker) throws
antlr.RecognitionException {
                /*
                 * redefine idt vector so as to store everything inside a new vector that will
                 * be stored inside the function definition, then we restore the original idt
vector
                 */
                AaslSymbolTable funcSymbolTable = new
AaslSymbolTable(walker.symt);
```

```
		// Add special undeclared types to the SymbolTable (func arguments)
		for(int i=0; i < fcallArgs.getNumArgs(); i++) {

			String varName =
thisFuncDef.functionPrototype.keys.elementAt(i);
			String varType =
thisFuncDef.functionPrototype.arg_list_decl.get(varName);
			AaslType aType;
			if (varType.compareTo("mixer") == 0) {
				throw new AaslException("Type argument not supported in
user defined function declaration "+
						this.functionName+"(..."+varType+"
"+varName+" ...)");
			} else if (varType.compareTo("osc") == 0) {
				throw new AaslException("Type argument not supported in
user defined function declaration "+
						this.functionName+"(..."+varType+"
"+varName+" ...)");
			} else if (varType.compareTo("oscbank") ==0) {
				throw new AaslException("Type argument not supported in
user defined function declaration "+
						this.functionName+"(..."+varType+"
"+varName+" ...)");
			} else if (varType.compareTo("int") == 0) {
				aType = new AaslInt(fcallArgs.args.elementAt(i));
			} else if (varType.compareTo("float") == 0) {
				aType = new AaslFloat(fcallArgs.args.elementAt(i));
			} else
				throw new RecognitionException("Unknown Variable type
equal to "+varType+
						" for function definition
"+thisFuncDef.toString() +
						" called by "+this.toString());

			funcSymbolTable.add( varName, aType);
		}

		/*
		 * Replace the system vector <v> and SymbolTable
		 */
		AaslSymbolTable oldSymbolTable = walker.symt;
		walker.symt = funcSymbolTable;
		Vector oldSystemVector = walker.v;
		walker.v = new Vector();

		/*
```

```
                 * Execute statements
                 */
        AaslType r = walker.expr( this.thisFuncDef.fbodyDef );
        r = walker.expr( this.thisFuncDef.fbodyCon );
        r = walker.expr( this.thisFuncDef.fbodyOut );
                Vector thisFuncV = walker.v;

        /*
         * Restore the system vector <v> and SymbolTable
         */
        walker.symt = oldSymbolTable;
        walker.v    = oldSystemVector;

        AaslExecutedFunction execFunc = new
AaslExecutedFunction(functionName+thisFuncDef.getTimesCalled(),
                fcallArgs, thisFuncDef, thisFuncV, walker.seg_arr.length); //why include
all this??
        walker.v.add( execFunc );

                return execFunc;
        }

        public String getType(){
                return "function call";
        }

        public void error(String msg){
                throw new AaslException( "Illegal operation \"" + msg + "\" : " +
                        getType() + "," + (name!=null ? name: "<null name>"));
        }

        public void error(AaslType b, String msg){
                if(b==null){
                        error(msg);
                        return;
                }
                throw new AaslException( "Illegal operation: \"" + msg + "\" : " +
                        getType() + "," + (name!=null ? name: "<null name> & ")
+
                        b.getType() + "," + (b.name!=null ? b.name: "<null
name>"));
        }

        public String toString() {
                StringBuffer output = new StringBuffer();
```

```
		output.append("<");
		output.append(functionName);
		output.append(fcallArgs.toString());
		output.append(">");

		return output.toString();
	}
}
```

AaslFunctionCallArguments.java

```
/*
 * By Carlos R. Perez,
 */
package aasl.intermediateTypes;

import java.util.Vector;

import antlr.collections.AST;


/**
 * Wrapper class for a function call
 */
public class AaslFunctionCallArguments extends AaslType
{
	/* Vector<String> */
	Vector<String> args;

	public AaslFunctionCallArguments() {
		args = new Vector<String>();
	}



	public AaslFunctionCallArguments(AST argList) {
		this.args = new Vector<String>();

		for(AST ptr=argList; ptr!=null; ptr = ptr.getNextSibling())
			this.addArg(ptr.getText());
	}

	public String getType(){
		return "function call arguments";
	}

	public int getNumArgs() {
		return args.size();
	}

	public String[] getArgs() {
		return (String[])args.toArray();
	}

	public void addArg(String arg) {
		args.add(arg);
	}

	public void error(String msg){
		throw new AaslException( "Illegal operation \"" + msg + "\" : " +
				getType() + "," + (name!=null ? name: "<null name>"));
	}

	public void error(AaslType b, String msg){
		if(b==null){
			error(msg);
			return;
		}
		throw new AaslException( "Illegal operation: \"" + msg + "\" : " +
				getType() + "," + (name!=null ? name: "<null name> & ")
+
				b.getType() + "," + (b.name!=null ? b.name: "<null
name>"));
	}

	public String toString() {
		StringBuffer output = new StringBuffer();

		output.append("(");
		for(int i=0; i < args.size(); i++) {
			if (args.size()-1 == i )
				output.append( (String)args.elementAt(i) );
			else
				output.append( (String)args.elementAt(i)+", " );
		}
		output.append(")");

		return output.toString();
	}

	public AaslFunctionCallArguments clone() {
		AaslFunctionCallArguments ret = new AaslFunctionCallArguments();
		ret.args = (Vector<String>)args.clone();
		return ret;
	}
}
```

AaslFunctionDef.java

```
/*
 * By Carlos R. Perez,
 */
package aasl.intermediateTypes;

import java.util.Vector;

import antlr.collections.AST;


/**
 * Wrapper class for a function call
 */
public class AaslFunctionDef extends AaslType
{
	String functionName;
	AaslFunctionDefArgList functionPrototype;
	AST fbodyDef;
	AST fbodyCon;
	AST fbodyOut;
	static private int timesCalled = 0;

	/*
	 * Add stuff to store the arguments whenever the function is called.
	 * So as whenever the function is called replace that with the true value.
	 */

	public AaslFunctionDef() {

	}

	public String getType(){
		return "function definition";
	}

	public void setFunctionName(String name) {
		functionName = name;
		this.name = functionName;
	}
	public String getFunctionName() {
		return functionName;
	}
	public int getTimesCalled(){
		return ++timesCalled;
	}
	public int getNumArgs() {
		return functionPrototype.getNumArgs();
	}

	public void setFunctionPrototype(AaslFunctionDefArgList prototype) {
		functionPrototype = prototype;
	}

	public void setASTFunctionBody( AST fbodyDef, AST fbodyCon, AST
fbodyOut ) {
		this.fbodyDef = fbodyDef;
		this.fbodyCon = fbodyCon;
		this.fbodyOut = fbodyOut;
	}


	public void error(String msg){
		throw new AaslException( "Illegal operation \"" + msg + "\" : " +
				getType() + "," + (name!=null ? name: "<null name>"));
	}

	public void error(AaslType b, String msg){
		if(b==null){
			error(msg);
			return;
		}
		throw new AaslException( "Illegal operation: \"" + msg + "\" : " +
				getType() + "," + (name!=null ? name: "<null name> & ")
+
				b.getType() + "," + (b.name!=null ? b.name: "<null
name>"));
	}

	public String toString() {
		StringBuffer output = new StringBuffer();
		output.append("<");
		output.append(functionName);
		output.append(functionPrototype.toString());
		output.append(">");
		return output.toString();
	}

	public AaslFunctionDef clone() {
		AaslFunctionDef ret = new AaslFunctionDef();
		ret.functionName = this.functionName;
		ret.functionPrototype = this.functionPrototype.clone();
		ret.fbodyDef = this.fbodyDef; //Reference, no need to copy it.
		ret.fbodyCon = this.fbodyCon;
		ret.fbodyOut = this.fbodyOut;
		ret.timesCalled = this.timesCalled;

		return ret;
	}
}
```

AaslFunctionDefArg.java

```
/*
 * By Carlos R. Perez,
 */
package aasl.intermediateTypes;

import java.util.Hashtable;
import java.util.Vector;

import antlr.collections.AST;


/**
 * Wrapper class for a function call
 */
public class AaslFunctionDefArgList extends AaslType
{
	Hashtable<String,String> arg_list_decl;
	Vector<String> keys;

	public AaslFunctionDefArgList() {
		arg_list_decl = new Hashtable<String,String>();
		keys = new Vector<String>();
	}

	public AaslFunctionDefArgList(AST argDef) {
		arg_list_decl = new Hashtable<String,String>();
		keys = new Vector<String>();
		String linenr = argDef.getLine()+":"+argDef.getColumn();
		int nr_args = 0;
		boolean voidDeclared = false;

		/* (type:ID argName:ID {
r.addArgDef(type.getText(),argName.getText()); } )* */
		for(AST ptr = argDef; ptr!=null && ptr.getNextSibling()!=null;
ptr=ptr.getNextSibling()) {
			++nr_args;
			String type = ptr.getText();
			ptr=ptr.getNextSibling();

			if (voidDeclared)
				throw new AaslException(linenr,"Semantic Function Error:
function already declared as void!");

			if ( (type.compareTo("void")==0) && (nr_args != 1) ) {
				throw new AaslException(linenr,"Semantic Function Error:
declaration is either 'void' only OR other type!");
			} else if ( (type.compareTo("void")==0) && (nr_args==1) ) {
				voidDeclared = true;
				this.addArgDef("void","void");
				continue;
			}

			String argName = ptr.getText();
			this.addArgDef(type,argName);
		}
	}

	public String getType(){
		return "function  definition argument list";
	}

	public boolean containsSymbol(String name) {
		for (int i=0; i<keys.size(); i++) {
			if ( keys.elementAt(i).compareTo(name) == 0 )
				return true;
		}

		return false;
	}

	public int getNumArgs() {
		return arg_list_decl.size();
	}
	public String getType(String varName) {
		return arg_list_decl.get(varName);
	}
	public String elementAt(int i) {
		return (String) keys.elementAt(i);
	}

	public void addArgDef(String type, String var_name) {
		arg_list_decl.put(var_name, type);
		keys.add(var_name);
	}

	public void error(String msg){
		throw new AaslException( "Illegal operation \"" + msg + "\" : " +
				getType() + "," + (name!=null ? name: "<null name>"));
	}

	public void error(AaslType b, String msg){
		if(b==null){
			error(msg);
			return;
		}
		throw new AaslException( "Illegal operation: \"" + msg + "\" : " +
				getType() + "," + (name!=null ? name: "<null name> & ")
+
				b.getType() + "," + (b.name!=null ? b.name: "<null
name>"));
	}

	public String toString() {
		StringBuffer output = new StringBuffer();

		output.append("(");
		for(int i=0; i < keys.size(); i++) {
//			Last output
			if (i+1 == keys.size()) {
				output.append(arg_list_decl.get(keys.elementAt(i)));
				output.append(" ");
				output.append(keys.elementAt(i));
			}
			else {
				output.append(arg_list_decl.get(keys.elementAt(i)));
				output.append(" ");
				output.append(keys.elementAt(i));
				output.append(", ");
			}
		}
		output.append(")");

		return output.toString();
	}


	public AaslFunctionDefArgList clone() {
		AaslFunctionDefArgList ret = new AaslFunctionDefArgList();
		ret.arg_list_decl = (Hashtable<String,String>)this.arg_list_decl.clone();
		ret.keys = (Vector<String>)this.keys.clone();

		return ret;
	}
}
```

AaslSegArray.java

```
/*
*	By Rob Katz,
*/

package aasl;

import aasl.intermediateTypes.AaslException;
import aasl.intermediateTypes.AaslType;

public class AaslSegArray extends AaslType {
	public AaslSegArray parent;
	boolean[] seg_arr;
	public int length;

	public AaslSegArray(int segs, AaslSegArray parent){
		this.parent = parent;
		this.seg_arr = new boolean[segs];
		this.length = segs;
		this.allFalse();
	}

	public void allFalse(){
		for(int i=0; i<length; i++){
			seg_arr[i]=false;
		}
	}

	public void allTrue(){
		for(int i=0; i<length; i++){
			seg_arr[i]=true;
		}
	}

	public boolean getValue(int index){
		return seg_arr[index];
	}

	public void setValue(int index){
		seg_arr[index] = true;
	}

	public int get_length(){
		return length;
	}

	public void invert(){
		for(int i=0; i<length; i++){
			seg_arr[i] = !seg_arr[i];
		}
	}

	/**
	 * Bit-wise OR with (this) and the argument.
	 * @param b
	 */
	public void merge(AaslSegArray b) {
		if (this.length != b.length)
			throw new AaslException("AaslSegArray.merge() failed: not the
same length!");

		for (int i=0; i<this.length; i++)
			this.seg_arr[i] = this.seg_arr[i] || b.seg_arr[i];
	}
}
```

AaslSymbolTable.java

```
/*
 * AaslSymbolTable.java
 * By  Rob Katz, Carlos R. Perez,
 *
 */
```

```
package aasl;

import java.util.*;

import aasl.intermediateTypes.*;

public class AaslSymbolTable extends HashMap{
	AaslSymbolTable parent;

	public AaslSymbolTable(AaslSymbolTable parent){
		this.parent = parent;
	}

	public AaslSymbolTable parent(){
		return parent;
	}

	public boolean containsSymbol(String name){
		AaslSymbolTable st = this;
		while(st != null)
		{
			if(st.containsKey(name))
				return true;
			else
				st = st.parent;
		}
		return false;
	}

	/**
	 * Only search on the current (top) symbol table.
	 *
	 * @param name
	 */
	public boolean topContainsSymbol(String name) {
		return this.containsKey(name);
	}

	public void add(String name, AaslType type){
		this.put(name, type);
	}

	public void setValue(String name, AaslType type){
		AaslSymbolTable st = this;
		while(st != null   &&   name != null)
		{
			if(st.containsSymbol(name)){
				this.put(name, type);
				return;
			}
			else
				st = st.parent;
		}
		throw new AaslException("Undeclared variable: " + type.getType() + " "
+ name);
	}

	public AaslType getValue(String name){
		AaslSymbolTable st = this;
		while(st != null)
		{
			if(st.containsKey(name))
				return (AaslType) st.get(name);
			else
				st = st.parent;
		}
		return null;
	}

}
```

AaslVectorUpdate.java

```
/*
   •  Vaishnav Janardhan
   */
package aasl;

import java.util.*;
import aasl.intermediateTypes.*;
public class AaslVectorUpdate
{
	Vector vect = new Vector();
	AaslVectorUpdate(Vector v)
	{
		this.vect = v;
	}

	AaslType getObject(String name)
	{
			for(int i=0;i<this.vect.size();i++)
			{
				if(name.equals(((AaslType)this.vect.elementAt(i)).name))
				{
					return (AaslType)this.vect.elementAt(i);
				}
			}
			return null;
		}


}
```

CodeGeneration.java

```
/*
*     By Rob Katz
*/

package aasl;

import synthEngine.SynthElement;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.*;

import javax.sound.sampled.AudioFileFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;

import junk.AaslMixer;

import synthEngine.Envelope;
import synthEngine.Oscillator;
import aasl.intermediateTypes.*;

/*
 * CodeGenerator's constructor is handed a vector.  The first element
of the vector
 * is a string representing the name of the output file. The second
element is
 * an AaslFloat representing the tone length. The third element is an
AaslInt
 * representing the number of segments.  The rest are AaslType Objects,
one
 * for each synth element declared.  Oscillator Banks are split up into
their
 * component oscillators.
 *
*/
```

```
public class CodeGenerator {
      /* Organize all data types, into arrays */
      Vector oscVec = new Vector();
      Vector mixVec = new Vector();
      Vector envVec = new Vector();
      Vector funcVector = new Vector();
      Vector outputVec = new Vector();
      String outputFile = new String();
      float tone_length;
      int segments;                                  /* total number
of segments */
      FileWriter outFile;
      PrintWriter out;
      FuncCodeGenerator fcg;

      boolean touched[][];    /* checks that there are no illegal loops
*/
      boolean oscDone[][];
      boolean mixDone[][];

      public CodeGenerator (Vector v){
            //check that the first two elements of v are correct
            if(!(v.elementAt(0) instanceof AaslString))
                  throw new AaslException("Bad intermediate
representation!");
            else outputFile = ((AaslString)(v.elementAt(0))).value;
            if(!(v.elementAt(1) instanceof AaslFloat))
                  throw new AaslException("Bad intermediate
representation!");
            else tone_length =
AaslType.floatValue((AaslFloat)v.elementAt(1));
            if(!(v.elementAt(2) instanceof AaslInt))
                  throw new AaslException("Bad intermediate
representation!");
            else segments=AaslInt.intValue((AaslInt)v.elementAt(2));

            if(!split_func(v))
                  throw new AaslException("Bad intermediate
represenation of functions!");
            //check that the rest are correct
            if(!vector_check(v, 3))
                  throw new AaslException("Bad intermediate
representation!");

            if(!cycle_check(v))
                  throw new AaslException("Design contains illegal
cycle!");

            /*
             * Split the Vectors Objects into their our CodeGen
             * Type Vectors (ex: oscVec)
             */
            if(!vector_split(v, 2))
                  throw new AaslException("Bad intermediate
representation!");
```

```
            for(int i=0; i<mixVec.size(); i++){
                  if( ((AaslMixer2)(mixVec.elementAt(i))).everOutput ){
                        AaslMixer2 mix = (AaslMixer2)(mixVec.remove(i--
));

                        outputVec.add(mix);
                  }
            }

            oscDone = new boolean[oscVec.size()][segments];
            mixDone = new boolean[mixVec.size()][segments];

            String codeFile = outputFile + ".java";
            try{
                  outFile = new FileWriter(codeFile);
            }
            catch(Exception ioe){
                  throw new AaslException("Failure in code
generation");
            }
            out = new PrintWriter(outFile);
            print_header(segments);
            out.println("");
            print_envelopes();
            out.println("");
            /*for(int i=0; i<funcVector.size(); i++){
                  AaslExecutedFunction aef =
(AaslExecutedFunction)funcVector.elementAt(i);
                  FuncCodeGenerator fcg = new
FuncCodeGenerator(setupFV(aef.getV()),out);
            }*/

            if(funcVector.size()!=0)
                  fcg = new FuncCodeGenerator(setupFV(funcVector),out);
            //print_functions();
            out.println("");
            print_elements();
            out.println("");
            print_outputs();
            out.println("");
            print_tail();
            out.close();

      }

      Vector setupFV(Vector v){
            v.add(0, new AaslFloat(tone_length));
            v.add(1, new AaslInt(segments));
            return v;
      }

      boolean split_func(Vector v){
            for(int i=0; i<v.size(); i++){
                  if(v.elementAt(i) instanceof AaslExecutedFunction){
                        funcVector.add(v.remove(i));
                        i--;
                  }
            }
            return true;
      }

      /*
       * vector_check takes a vector and an int representing the
element at which
       * the check begins.  It ensures that all elements of the vector
from i onward
       * are AaslTypes.  If it hits a mixer, it makes a recursive call
to ensure
       * that all elements in the mixers object vector are also
AaslTypes.
       */
      boolean vector_check(Vector v, int i){
            for(; i<v.size(); i++){
                  if(!(v.elementAt(i) instanceof AaslType)) return
false;
                  if(v.elementAt(i) instanceof AaslMixer2){
                        AaslMixer2 mix = (AaslMixer2)v.elementAt(i);
                        vector_check(mix.v, 0);
                  }
            }
            return true;
      }//end vector_check

      /*
       * cycle_check takes the vector handed to CodeGeneration and
ensures there
       * are no cycles.  That is, we need to ensure that no oscillator
output is
       * indirectly attached to one of its inputs.
       */

      boolean cycle_check(Vector v){
            for(int i=0; i<segments; i++){
                  for(int j=0; j<v.size(); j++){
                        if(v.elementAt(j) instanceof AaslOscillator2){
                              AaslOscillator2 osc =
(AaslOscillator2)(v.elementAt(j));
                              /* sanity checks */
                              if (osc.freq[i] == null)
                                    throw new AaslException("Oscillator
"+osc.name+
                                                " frequency is
incompletely connected!");
                              if (osc.amp[i] == null)
                                    throw new AaslException("Oscillator
"+osc.name+
                                                " amplitude is not
defined!");
                              if( !(check_connect(osc, osc.freq[i], i)
&&
                                          check_connect(osc,
osc.amp[i], i)))
                                    return false;
                        }
                  }//end inner for
            }//end outer for
            return true;
      }

      /*
       * check_connect is a recursive helper function called by
cycle_check()
       */

      boolean check_connect(AaslType element, AaslType connection, int
seg){
            AaslType root = element;
            if( (connection instanceof AaslFloat) || (connection
instanceof AaslInt)
                        || (connection instanceof AaslEnvelope))
                  return true;
            if(connection == root)
                  return false;
            if(connection instanceof AaslOscillator2){
                  AaslOscillator2 osc=((AaslOscillator2)(connection));
                  return ( check_connect(root, osc.freq[seg], seg) &&
                               check_connect(root, osc.amp[seg], seg)
);
            }
            if(connection instanceof AaslMixer2){
                  AaslMixer2 mix = (AaslMixer2)connection;
                  for(int i=0; i<mix.v.size(); i++){
                        if(!check_connect(root, (AaslType)
mix.v.elementAt(i), seg))
                              return false;
                  }
                  return true;
            }
            return false;
      }

      /*
       * Split the Vectors Objects into their our CodeGen
       * Type Vectors (ex: oscVec), also check the integrety of the
Vector, in
       * that all elements must be at least of subclass AaslType.
       */
      boolean vector_split(Vector v, int start){
            for(int i=start; i<v.size(); i++){
                  /* All objects in Vector must be of AaslType */
                  if (!(v.elementAt(i) instanceof AaslType))
                        return false;

                  AaslType thisElement = (AaslType)(v.elementAt(i));
                  if(thisElement instanceof AaslOscillator2){
                        boolean add = true;
                        for(int j=0; j<oscVec.size(); j++){
                              if(
((AaslType)oscVec.elementAt(j)).name.compareTo(thisElement.name) == 0){
                                    add=false;
                              }
                        }
                        if(add)
                              oscVec.add(thisElement);
                  }
                  else if(thisElement instanceof AaslMixer2){
                        vector_split( ((AaslMixer2)(thisElement)).v,
0);
                        if(!(mixVec.contains(thisElement)))
                              mixVec.add(thisElement);
                  }
                  else if(thisElement instanceof AaslEnvelope)
                        envVec.add(thisElement);
            }
            return true;
      }//end vector_split

      void print_header(int segments){
            out.println("import javax.sound.sampled.AudioFormat;");
            out.println("import javax.sound.sampled.AudioFileFormat;");
            out.println("import javax.sound.sampled.AudioSystem;");
            out.println("import javax.sound.sampled.DataLine;");
            out.println("import javax.sound.sampled.TargetDataLine;");
            out.println("import
javax.sound.sampled.AudioInputStream;");
            out.println("import java.io.*;");
            out.println("");
            out.println("import synthEngine.*;");
            out.println("");
            out.println("public class " + outputFile + "{");
            out.println("\tprivate static double MAXAMP = 32760;");
            out.println("\tprivate static boolean ENVELOPE = true;");
            out.println("\tprivate static boolean OSCILLATOR =
false;");
            out.println("\tprivate static TargetDataLine line =
null;");
            out.println("\tprivate static AudioFormat format = null;");
            out.println("\tprivate static AudioInputStream ais =
null;");
            out.println("\tprivate static final double tone_length = "
+ tone_length + ";");
            out.println("\tprivate static final String FILENAME = \"" +
outputFile + ".wav" + "\";");
            out.println("\tprivate static final int num_samples =(int)
" +
                        "(tone_length * SynthElement.SAMPRATE);");
            out.println("\tprivate static byte[] samples = new
byte[num_samples * 2];");
            out.println("\tprivate static int segments = " + segments +
";");
            out.println("");
            out.println("\tpublic static void main(String args[]){");

            out.println("\t\tformat =  new
AudioFormat(AudioFormat.Encoding.PCM_SIGNED,");
            out.println("\t\t\tSynthElement.SAMPRATE, 16, 1, 2,
SynthElement.SAMPRATE, false);");
            out.println("\t\ttry {");
            out.println("\t\t\tDataLine.Info info = new
DataLine.Info(TargetDataLine.class, format);");
            out.println("\t\t\tif (!AudioSystem.isLineSupported(info))
{");
            out.println("\t\t\t\tSystem.out.println(\"Line does not
support: \" + format);");
            out.println("\t\t\t\tSystem.exit(0);");
            out.println("\t\t\t}");
            out.println("\t\t\tline = (TargetDataLine)
AudioSystem.getLine(info);");
            out.println("\t\t}");
            out.println("\t\tcatch (Exception e)");
            out.println("\t\t{  System.out.println( e.getMessage(
));");
            out.println("\t\tSystem.exit(0);");
            out.println("\t\t}");
      }//end print_header()

      void print_envelopes(){
            for(int i=0; i<envVec.size(); i++){
                  AaslEnvelope thisEnv =
(AaslEnvelope)(envVec.elementAt(i));
                  out.println("//Envelope " + thisEnv.name);
                  out.println("double[][] " + thisEnv.name + "array =
new double["
                                   + thisEnv.v.size() + "][2];");
                  for(int j=0; j<thisEnv.v.size(); j++){
                        AaslEnvelopePair ep =
(AaslEnvelopePair)(thisEnv.v.elementAt(j));
                        out.print(thisEnv.name + "array[" + j + "][0] =
" + ep.time + ";   ");
                        out.println(thisEnv.name + "array[" + j + "][1]
= " + ep.value + ";");
                  }//end inner for
                  out.println("Envelope " + thisEnv.name + " = new
Envelope("
                              + thisEnv.name + "array, " +
thisEnv.v.size() +
                              ", tone_length);");
                  out.println();
            }
      }//end print_envelopes

      void print_functions(){
            if(!vector_check(funcVector, 0))
                  throw new AaslException("Incorrect intermediate
representation of function.");
            if(!cycle_check(funcVector))
                  throw new AaslException("Incorrect intermediate
representation of function.");

      }

      void print_elements(){
            boolean added = true;
            for(int i=0; i<oscVec.size(); i++){
                  for(int j=0; j<segments; j++)
```

```
                        oscDone[i][j]=false;
                }
                for(int i=0; i<mixVec.size(); i++){
                        for(int j=0; j<segments; j++)
                                mixDone[i][j]=false;
                }
        //take care of all osc with constant inputs
                for(int j=0; j<segments; j++){
                        added=true;
                        while(added){
                                added = false;
                                //first go through all the oscillators
                                for(int i=0; i<oscVec.size(); i++){
                                        if( (oscDone[i][j] == false)){
                                                AaslOscillator2 osc =
(AaslOscillator2)(oscVec.elementAt(i));
                                                AaslType freqCon = osc.freq[j];
                                                AaslType ampCon = osc.amp[j];
                                                if( ((freqCon instanceof AaslFloat)
|| (freqCon instanceof AaslInt))
                                                        &&((ampCon instanceof
AaslFloat) || (ampCon instanceof AaslInt)) ){

        out.println("//"+oscVec.size());
                                                        print_osc(osc, j, true,
true);
                                                        added=true;
                                                        oscDone[i][j]=true;
                                                }

                                                else if((ampCon instanceof
AaslFloat) || (ampCon instanceof AaslInt)){
                                                        int
freqConIndex=getIndex(freqCon);
                                                        if(freqCon instanceof
AaslOscillator2 &&

        oscDone[freqConIndex][j]){
                                                                print_osc(osc, j,
false, true);
                                                                added=true;
                                                                oscDone[i][j]=true;
                                                        }//end if

                                                        else if(freqCon instanceof
AaslMixer2 &&

        mixDone[freqConIndex][j]){
                                                                print_osc(osc, j,
false, true);
                                                                added=true;
                                                                oscDone[i][j]=true;
                                                        }//end else if
                                                        else if(freqCon instanceof
AaslEnvelope ||
                                                                        freqCon
instanceof AaslExecutedFunction){
```

```
                                                print_osc(osc, j,
false, true);

                                                added=true;
                                                oscDone[i][j]=true;
                                          }//end else if

                                    }//end else if

                                    else if((freqCon instanceof
AaslFloat) || (freqCon instanceof AaslInt)){
                                          int
ampConIndex=getIndex(ampCon);
                                          if(ampCon instanceof
AaslOscillator2 &&

      oscDone[ampConIndex][j]){

      print_osc(osc,j,true,false);
                                                added=true;
                                                oscDone[i][j]=true;
                                          }
                                          else if(ampCon instanceof
AaslMixer2 &&

      mixDone[ampConIndex][j]==true){

      print_osc(osc,j,true,false);
                                                added=true;
                                                oscDone[i][j]=true;
                                          }
                                          else if(ampCon instanceof
AaslEnvelope ||
                                                      ampCon instanceof
AaslExecutedFunction){
                                                print_osc(osc, j, true,
false);
                                                added=true;
                                                oscDone[i][j]=true;
                                          }
                                    }//end else if

                                    else//if both are variable
                                    {
                                          boolean freqDone=false;
                                          boolean ampDone=false;
                                          int
freqConIndex=getIndex(freqCon);
                                          int
ampConIndex=getIndex(ampCon);

                                          if(freqCon instanceof
AaslEnvelope ||
                                                      freqCon
instanceof AaslExecutedFunction)
                                                freqDone=true;
                                          else if(freqCon instanceof
AaslOscillator2)
```

```
      freqDone=oscDone[freqConIndex][j];
                                    else if(freqCon instanceof
AaslMixer2)

      freqDone=mixDone[freqConIndex][j];

                                    if(ampCon instanceof
AaslEnvelope ||
                                                ampCon instanceof
AaslExecutedFunction)
                                          ampDone=true;
                                    if(ampCon instanceof
AaslOscillator2)

      ampDone=oscDone[ampConIndex][j];
                                    if(ampCon instanceof
AaslMixer2)

      ampDone=mixDone[ampConIndex][j];

                                    if(freqDone && ampDone){
                                          print_osc(osc, j,
false, false);

                                          added=true;
                                          oscDone[i][j]=true;
                                    }
                              }//end else
                        }
                  }//end inner for
                  for(int i=0; i<mixVec.size(); i++){
                        AaslMixer2 mix =
(AaslMixer2)(mixVec.elementAt(i));
                        boolean mixReady=true;
                        if(mixDone[getIndex(mix)][j]) continue;
                        for(int k=0; k<mix.v.size(); k++){
                              AaslType element=
(AaslType)mix.v.elementAt(k);
                              if(element instanceof
AaslOscillator2){

      if(!oscDone[getIndex(element)][j])
                                          mixReady=false;
                              }
                              else if(element instanceof
AaslMixer2){

      if(!mixDone[getIndex(element)][j])
                                          mixReady=false;
                              }
                        }//end inner inner for
                        if(mixReady){
                              print_mix(mix, j);
                              added=true;
                              mixDone[i][j]=true;
                        }
                  }//end inner for
                  }//end while
            }//end outer for
      }//end print_elements



      void print_osc(AaslOscillator2 osc, int seg, boolean
constantFreq,
                  boolean constantAmp){
            out.println("//print_osc");
            String condStr;
            float constFreqValue = 0;
            float constAmpValue = 0;
            out.print("Oscillator " + osc.name + seg + " = new
Oscillator(");
            condStr = constantFreq ? ""
                                          : (osc.centralFreq[seg] + ", ");
            out.print(condStr + "\"" + osc.waveType[seg] + "\",
tone_length/segments, ");
            if(constantFreq){
                  constFreqValue =
(AaslType.floatValue(osc.freq[seg]));
                  out.print(constFreqValue);
            }
            else if(osc.freq[seg] instanceof AaslOscillator2 ||
                        osc.freq[seg] instanceof AaslMixer2)
                  out.print(((AaslType)osc.freq[seg]).name + seg +
".output");
            else if(osc.freq[seg] instanceof AaslEnvelope)
                  out.print(((AaslType)osc.freq[seg]).name + ".output("
+ seg + "," + segments + ")");
            else if(osc.freq[seg] instanceof AaslExecutedFunction)
                  out.print( ((AaslExecutedFunction)osc.freq[seg]).name
+ "OUTPUT$$" + seg);

            if(constantAmp){
                  constAmpValue = (AaslType.floatValue(osc.amp[seg]));
                  out.println(", " + constAmpValue + ");");
            }
            else if(osc.amp[seg] instanceof AaslOscillator2 ||
                        osc.amp[seg] instanceof AaslMixer2){
                  out.print(", " + ((AaslType)osc.amp[seg]).name + seg
+".output, false);");
            }
            else if(osc.amp[seg] instanceof AaslEnvelope)
                  out.print(", " + ((AaslType)osc.amp[seg]).name
+".output(" + seg + "," + segments + "), true);");
            else if(osc.amp[seg] instanceof AaslExecutedFunction)
                  out.println(
((AaslExecutedFunction)osc.amp[seg]).name + "OUTPUT$$" +
                              seg + ", false);");
      }//end print_osc()


      //mixer outputs are arrays of doubles named name$seg
      void print_mix(AaslMixer2 mix, int seg){
            out.println("//print_mix()");
            out.println("double[] "+ mix.name + "OUTPUT$" + seg +
                        " = new double[num_samples/segments];");
            out.println("for(int i=0; i<(num_samples/segments);" +
                        " i++){");
            for(int j=0; j<mix.numElements(); j++){
                  out.print("\t\t" + mix.name + "OUTPUT$" +seg + "[i]
+= ");
                  AaslType mixElement = (AaslType)mix.v.elementAt(j);
                  if(mixElement instanceof AaslOscillator2){
                        out.println(mixElement.name + seg + ".output[i]
* " +

      ((AaslOscillator2)(mixElement)).factor[seg] + ";");
                  }
                  else if(mixElement instanceof AaslMixer2){
                        out.println(mixElement.name +"OUTPUT$" + seg +
"[i] * " +

      ((AaslMixer2)(mixElement)).factor[seg] +";");
                  }
                  else if(mixElement instanceof AaslExecutedFunction){
                        out.println(mixElement.name + "OUTPUT$" + seg +
"[i] * " +

      ((AaslExecutedFunction)mixElement).factor[seg] + ";");
                  }
            }
            out.println("\t\t" + mix.name + "OUTPUT$" + seg +"[i] /= "
+
                        mix.factorSum[seg] + ";");
            out.println("}");
      }//end print_mix

      void print_outputs(){
            check_outputs();
            for(int i=0; i<segments; i++){
                  for(int j=0; j<outputVec.size(); j++){
                        AaslMixer2 mix =
(AaslMixer2)(outputVec.elementAt(j));
                        if(mix.output[i]){
                              out.println("double[] OUTPUT$$" + i +
                                          " = new
double[num_samples/segments];");
                              out.println("for(int i=0;
i<num_samples/segments; i++){");
                              for(int k=0; k<mix.numElements(); k++){
                                    out.print("\t\tOUTPUT$$" + i + "[i]
+= ");
                                    AaslType mixElement =
(AaslType)mix.v.elementAt(k);
                                    if(mixElement instanceof
AaslOscillator2){
                                          out.println(mixElement.name +
i + ".output[i] * " +

      ((AaslOscillator2)(mixElement)).factor[i] + ";");
                                    }
                                    else if(mixElement instanceof
AaslMixer2){
                                          out.println(mixElement.name +
"OUTPUT$" + i + "[i] * " +

      ((AaslMixer2)(mixElement)).factor[i] +";");
                                    }
                                    else if(mixElement instanceof
AaslExecutedFunction){
                                          out.println(mixElement.name +
"OUTPUT$$" + i + "[i] * " +
                                                      "1.0;");

      //((AaslExecutedFunction)mixElement).factor[i] + ";");
                                    }
                              }
                              out.println("\t\tOUTPUT$$" + i +"[i] /= "
+
                                          mix.factorSum[i] + ";");
                              out.println("}");
                        }
                  }
            }
      }

      void check_outputs(){
            boolean[] segs = new boolean[segments];
            for(int i=0; i<segments; i++) {segs[i] = false;}
            for(int i=0; i<outputVec.size(); i++){
                  for(int j=0; j<segments; j++){
                        if(
((AaslMixer2)(outputVec.elementAt(i))).output[j] ){
                              if(!segs[j])
                                    segs[j]=true;
                              else
                                    throw new AaslException("Bad Output
Assignment.  More " +
                                                "than one mixer
assigned in segment" + j);
                        }
                  }
            }
            for(int i=0; i<segments; i++){
                  if(segs[i] == false)
                        throw new AaslException("Bad Output Assignment.
One or more" +
                                    " segments has no output
assigned!");
            }
      }

      int getIndex(AaslType element){
            if(element instanceof AaslOscillator2){
                  for(int i=0; i<oscVec.size(); i++){
                        if( element.name.equals(
((AaslOscillator2)(oscVec.elementAt(i))).name ) )
                              return i;
```

```
				}
			}
			else if(element instanceof AaslMixer2){
				for(int i=0; i<mixVec.size(); i++){
					if( element.name.equals(
((AaslMixer2)(mixVec.elementAt(i))).name ) )
						return i;
				}
			}
			return -1;
		}

		void print_tail(){
			out.println("double[][] outputArray = new double[" +
segments
					+ "]" + "[(int)(num_samples/segments)];");
			//change made here
			for(int i=0; i<segments; i++){
				out.println("outputArray[" + i + "] = OUTPUT$$" + i +
";");
			}
			out.println("for(int i=0; i<segments; i++){");
			out.println("\tfor(int j=0, index=0;
j<num_samples/segments;" +
					" j++, index+=2){");
			out.println("\t\tint sample = (int)(MAXAMP *
outputArray[i][j]);");

		out.println("\t\tsamples[(int)(2*i*((int)num_samples/segments)) +
index] = " +
					"(byte)(sample & 0xFF);");

		out.println("\t\tsamples[(int)(2*i*((int)num_samples/segments)) +
index + 1] = " +
			"(byte)(sample >>8 & 0xFF);");
			out.println("\t}");
			out.println("}");
			out.println("ais = new AudioInputStream(new
ByteArrayInputStream(samples),");
			out.println("\t\t\tformat, samples.length);");
			out.println("try{");
			out.println("\tAudioSystem.write(ais,
AudioFileFormat.Type.WAVE," +
					" new File(FILENAME));");
			out.println("}");
			out.println("catch(Exception e){");
			out.println("\tSystem.out.println(e);");
			out.println("}");
			out.println("System.exit(0);");
			out.println("}");
			out.println("}");


		}
}
```

FunctionCodeGeneration.java

```
/*
*	By Rob Katz,
*/

package aasl;

import synthEngine.SynthElement;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.*;

import javax.sound.sampled.AudioFileFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;

import junk.AaslMixer;

import synthEngine.Envelope;
import synthEngine.Oscillator;
import aasl.intermediateTypes.*;

/*
 * CodeGenerator's constructor is handed a vector.  The first element of the vector
 * is a string representing the name of the output file. The second element is
 * an AaslFloat representing the tone length. The third element is an AaslInt
 * representing the number of elements.  The rest are AaslType Objects, one
 * for each synth element declared.  Oscillator Banks are split up into their
 * component oscillators.
 *
*/

public class FuncCodeGenerator {
	/* Organize all data types, into arrays */
	Vector oscVec = new Vector();
	Vector mixVec = new Vector();
	Vector envVec = new Vector();
	Vector outputVec = new Vector();
	Vector curVec = new Vector();
	AaslExecutedFunction aef;
	float tone_length;
	int segments;					/* total number of segments 
*/
	FileWriter outFile;
	PrintWriter out;

	boolean touched[][];	/* checks that there are no illegal loops */
	boolean oscDone[][];
	boolean mixDone[][];

	public FuncCodeGenerator (Vector v, PrintWriter outFile){
		//check that the first two elements of v are correct
		if(!(v.elementAt(0) instanceof AaslFloat))
			throw new AaslException("Bad intermediate representation!");
		else tone_length = AaslType.floatValue((AaslFloat)v.elementAt(0));
		if(!(v.elementAt(1) instanceof AaslInt))
			throw new AaslException("Bad intermediate representation!");
		else segments=AaslInt.intValue((AaslInt)v.elementAt(1));

		for(int i=2; i<v.size(); i++){
			oscVec = new Vector();
			mixVec = new Vector();
			envVec = new Vector();
			outputVec = new Vector();
			aef = (AaslExecutedFunction)(v.elementAt(i));
			curVec = aef.getV();
			if(!vector_check(curVec, 0))
				throw new AaslException("Bad intermediate 
representation!");
			if(!cycle_check(curVec))
				throw new AaslException("Design contains illegal 
cycle!");

		/*
		 * Split the Vectors Objects into their our CodeGen
		 * Type Vectors (ex: oscVec)
		 */

			if(!vector_split(curVec, 0))
				throw new AaslException("Bad intermediate 
representation!");
			for(int j=0; j<mixVec.size(); j++){
				if( ((AaslMixer2)(mixVec.elementAt(j))).everOutput ){
					AaslMixer2 mix = (AaslMixer2)(mixVec.remove(j-
-));

					outputVec.add(mix);
				}
				}

				oscDone = new boolean[oscVec.size()][segments];
				mixDone = new boolean[mixVec.size()][segments];

				out = outFile;
				print_envelopes();
				out.println("");
				print_elements();
				out.println("");
				print_outputs();
			}
		}

		/*boolean split_func(Vector v){
			for(int i=0; i<v.size(); i++){
				if(v.elementAt(i) instanceof AaslExecutedFunction){
					funcVector.add(v.remove(i));
					i--;
				}
			}
			return true;
		}*/

		/*
		 * vector_check takes a vector and an int representing the element at which
		 * the check begins.  It ensures that all elements of the vector from i onward
		 * are AaslTypes.  If it hits a mixer, it makes a recursive call to ensure
		 * that all elements in the mixers object vector are also AaslTypes.
		 */
		boolean vector_check(Vector v, int i){
			for(; i<v.size(); i++){
				if(!(v.elementAt(i) instanceof AaslType)) return false;
				if(v.elementAt(i) instanceof AaslMixer2){
					AaslMixer2 mix = (AaslMixer2)v.elementAt(i);
					vector_check(mix.v, 0);
				}
			}
			return true;
		}//end vector_check

		/*
		 * cycle_check takes the vector handed to CodeGeneration and ensures there
		 * are no cycles.  That is, we need to ensure that no oscillator output is
		 * indirectly attached to one of its inputs.
		 */
```

```
	boolean cycle_check(Vector v){
		for(int i=0; i<segments; i++){
			for(int j=0; j<v.size(); j++){
				if(v.elementAt(j) instanceof AaslOscillator2){
					AaslOscillator2 osc =
(AaslOscillator2)(v.elementAt(j));
					if( !(check_connect(osc, osc.freq[i], i) &&
							check_connect(osc, osc.amp[i], i)))
						return false;
				}
			}//end inner for
		}//end outer for
		return true;
	}

	/*
	 * check_connect is a recursive helper function called by cycle_check()
	 */

	boolean check_connect(AaslType element, AaslType connection, int seg){
		AaslType root = element;
		if( (connection instanceof AaslFloat) || (connection instanceof AaslInt)
				|| (connection instanceof AaslEnvelope))
			return true;
		if(connection == root)
			return false;
		if(connection instanceof AaslOscillator2){
			AaslOscillator2 osc=((AaslOscillator2)(connection));
			return ( check_connect(root, osc.freq[seg], seg) &&
					check_connect(root, osc.amp[seg], seg) );
		}
		if(connection instanceof AaslMixer2){
			AaslMixer2 mix = (AaslMixer2)connection;
			for(int i=0; i<mix.v.size(); i++){
				if(!check_connect(root, (AaslType) mix.v.elementAt(i),
seg))
					return false;
			}
			return true;
		}
		return false;
	}

	/*
	 * Split the Vectors Objects into their our CodeGen
	 * Type Vectors (ex: oscVec), also check the integrety of the Vector, in
	 * that all elements must be at least of subclass AaslType.
	 */
	boolean vector_split(Vector v, int start){
		for(int i=start; i<v.size(); i++){
			/* All objects in Vector must be of AaslType */
			if (!(v.elementAt(i) instanceof AaslType))
				return false;

			AaslType thisElement = (AaslType)(v.elementAt(i));
			if(thisElement instanceof AaslOscillator2){
				boolean add=true;
				for(int j=0; j<oscVec.size(); j++){
					if(
((AaslOscillator2)(oscVec.elementAt(j))).name.compareTo(
							thisElement.name) == 0){
						add=false;
					}
				}
				if(add)
					oscVec.add(thisElement);
			}
			else if(thisElement instanceof AaslMixer2){
				vector_split( ((AaslMixer2)(thisElement)).v, 0);
				if(!(mixVec.contains(thisElement)))
					mixVec.add(thisElement);
			}
			else if(thisElement instanceof AaslEnvelope)
				envVec.add(thisElement);
		}
		return true;
	}//end vector_split



	void print_envelopes(){
		for(int i=0; i<envVec.size(); i++){
			AaslEnvelope thisEnv = (AaslEnvelope)(envVec.elementAt(i));
			out.println("//Envelope " + thisEnv.name + "$" + aef.name);
			out.println("double[][] " + thisEnv.name + "$" + aef.name  +
"array = new double["
						+ thisEnv.v.size() + "][2];");
			for(int j=0; j<thisEnv.v.size(); j++){
				AaslEnvelopePair ep =
(AaslEnvelopePair)(thisEnv.v.elementAt(j));
				out.print(thisEnv.name + "$" + aef.name  + "array[" + j +
"][0] = " + ep.time + ";   ");
```

```
                                                out.println(thisEnv.name + "$" + aef.name  + "array[" + j +
"][1] = " + ep.value + ";");
                                }//end inner for
                                out.println("Envelope " + thisEnv.name + "$" + aef.name  + " =
new Envelope("
                                                        + thisEnv.name + "$" + aef.name  + "array, " +
thisEnv.v.size() +
                                                        ", tone_length);");
                                out.println();
                        }
                }//end print_envelopes


                void print_elements(){
                        boolean added = true;
                        for(int i=0; i<oscVec.size(); i++){
                                for(int j=0; j<segments; j++)
                                        oscDone[i][j]=false;
                        }
                        for(int i=0; i<mixVec.size(); i++){
                                for(int j=0; j<segments; j++)
                                        mixDone[i][j]=false;
                        }
                //take care of all osc with constant inputs
                        for(int j=0; j<segments; j++){
                                added=true;
                                while(added){
                                        added = false;
                                        //first go through all the oscillators
                                        for(int i=0; i<oscVec.size(); i++){
                                                if( (oscDone[i][j] == false)){
                                                        AaslOscillator2 osc =
(AaslOscillator2)(oscVec.elementAt(i));
                                                        AaslType freqCon = osc.freq[j];
                                                        AaslType ampCon = osc.amp[j];
                                                        if( ((freqCon instanceof AaslFloat) ||
(freqCon instanceof AaslInt))
                                                                &&((ampCon instanceof AaslFloat)
|| (ampCon instanceof AaslInt)) ){
                                                                out.println("//"+oscVec.size());
                                                                print_osc(osc, j, true, true);
                                                                added=true;
                                                                oscDone[i][j]=true;
                                                        }
                                        else if((ampCon instanceof AaslFloat) ||
(ampCon instanceof AaslInt)){
                                             int
freqConIndex=getIndex(freqCon);
                                             if(freqCon instanceof
AaslOscillator2 &&

     oscDone[freqConIndex][j]){
                                                  print_osc(osc, j, false, true);
                                                  added=true;
                                                  oscDone[i][j]=true;
                                             }//end if

                                             else if(freqCon instanceof
AaslMixer2 &&

     mixDone[freqConIndex][j]){
                                                  print_osc(osc, j, false, true);
                                                  added=true;
                                                  oscDone[i][j]=true;
                                             }//end else if
                                             else if(freqCon instanceof
AaslEnvelope ||
                                                       freqCon instanceof
AaslExecutedFunction){
                                                  print_osc(osc, j, false, true);
                                                  added=true;
                                                  oscDone[i][j]=true;
                                             }//end else if

                                        }//end else if

                                        else if((freqCon instanceof AaslFloat) ||
(freqCon instanceof AaslInt)){
                                             int
ampConIndex=getIndex(ampCon);
                                             if(ampCon instanceof
AaslOscillator2 &&

     oscDone[ampConIndex][j]){
                                                  print_osc(osc,j,true,false);
                                                  added=true;
                                                  oscDone[i][j]=true;
                                             }
                                             else if(ampCon instanceof
AaslMixer2 &&
	mixDone[ampConIndex][j]==true){
							print_osc(osc,j,true,false);
							added=true;
							oscDone[i][j]=true;
						}
						else if(ampCon instanceof
AaslEnvelope ||
								ampCon instanceof
AaslExecutedFunction){
							print_osc(osc, j, true, false);
							added=true;
							oscDone[i][j]=true;
						}
					}//end else if

					else//if both are variable
					{
						boolean freqDone=false;
						boolean ampDone=false;
						int
freqConIndex=getIndex(freqCon);
						int
ampConIndex=getIndex(ampCon);

						if(freqCon instanceof AaslEnvelope
||
								freqCon instanceof
AaslExecutedFunction)
							freqDone=true;
						else if(freqCon instanceof
AaslOscillator2)

	freqDone=oscDone[freqConIndex][j];
						else if(freqCon instanceof
AaslMixer2)

	freqDone=mixDone[freqConIndex][j];

						if(ampCon instanceof AaslEnvelope
||
								ampCon instanceof
AaslExecutedFunction)
							ampDone=true;
						if(ampCon instanceof
AaslOscillator2)
```

```
	ampDone=oscDone[ampConIndex][j];
						if(ampCon instanceof AaslMixer2)

	ampDone=mixDone[ampConIndex][j];

						if(freqDone && ampDone){
							print_osc(osc, j, false, false);
							added=true;
							oscDone[i][j]=true;
						}
					}//end else
				}
			}//end inner for
			for(int i=0; i<mixVec.size(); i++){
				AaslMixer2 mix =
(AaslMixer2)(mixVec.elementAt(i));
				boolean mixReady=true;
				if(mixDone[getIndex(mix)][j]) continue;
				for(int k=0; k<mix.v.size(); k++){
					AaslType element=
(AaslType)mix.v.elementAt(k);
					if(element instanceof AaslOscillator2){
						if(!oscDone[getIndex(element)][j])
							mixReady=false;
					}
					else if(element instanceof AaslMixer2){
						if(!mixDone[getIndex(element)][j])
							mixReady=false;
					}
				}//end inner inner for
				if(mixReady){
					print_mix(mix, j);
					added=true;
					mixDone[i][j]=true;
				}
			}//end inner for
		}//end while
	}//end outer for
}//end print_elements



void print_osc(AaslOscillator2 osc, int seg, boolean constantFreq,
		boolean constantAmp){
	out.println("//print_osc");
```

```
		String condStr;
		float constFreqValue = 0;
		float constAmpValue = 0;
		out.print("Oscillator " + osc.name + "$" + aef.name + seg + " = new
Oscillator(");
		condStr = constantFreq ? ""
					: (osc.centralFreq[seg] + ", ");
		out.print(condStr + "\"" + osc.waveType[seg] + "\", tone_length/segments,
");
		if(constantFreq){
			constFreqValue = (AaslType.floatValue(osc.freq[seg]));
			out.print(constFreqValue);
		}
		else if(osc.freq[seg] instanceof AaslOscillator2 ||
				osc.freq[seg] instanceof AaslMixer2)
			out.print(((AaslType)osc.freq[seg]).name + seg + ".output");
		else if(osc.freq[seg] instanceof AaslEnvelope)
			out.print(((AaslType)osc.freq[seg]).name + ".output(" + seg + "," +
segments + ")");
		else if(osc.freq[seg] instanceof AaslExecutedFunction)
			out.print( ((AaslExecutedFunction)osc.freq[seg]).name +
"OUTPUT$$" + seg);

		if(constantAmp){
			constAmpValue = (AaslType.floatValue(osc.amp[seg]));
			out.println(", " + constAmpValue + ");");
		}
		else if(osc.amp[seg] instanceof AaslOscillator2 ||
				osc.amp[seg] instanceof AaslMixer2){
			out.print(", " + ((AaslType)osc.amp[seg]).name + seg +".output,
false);");
		}
		else if(osc.amp[seg] instanceof AaslEnvelope)
			out.print(", " + ((AaslType)osc.amp[seg]).name +".output(" + seg
+ "," + segments + "), true);");
		else if(osc.amp[seg] instanceof AaslExecutedFunction)
			out.println( ((AaslExecutedFunction)osc.amp[seg]).name +
"OUTPUT$$" +
					seg + ", false);");
	}//end print_osc()


	//mixer outputs are arrays of doubles named name$seg
	void print_mix(AaslMixer2 mix, int seg){
		out.println("//print_mix()");
		out.println("double[] "+ mix.name + "$" + aef.name  + "OUTPUT$" + seg
+
				" = new double[num_samples/segments];");
		out.println("for(int i=0; i<(num_samples/segments);" +
				" i++){");
		for(int j=0; j<mix.numElements(); j++){
			out.print("\t\t" + mix.name  + "$" + aef.name + "OUTPUT$" +seg
+ "[i] += ");
			AaslType mixElement = (AaslType)mix.v.elementAt(j);
			if(mixElement instanceof AaslOscillator2){
				out.println(mixElement.name  + "$" + aef.name + seg +
".output[i] * " +
						((AaslOscillator2)(mixElement)).factor[seg]
+ ";");
			}
			else if(mixElement instanceof AaslMixer2){
				out.println(mixElement.name  + "$" + aef.name
+"OUTPUT$" + seg + "[i] * " +
						((AaslMixer2)(mixElement)).factor[seg]
+";");
			}
			else if(mixElement instanceof AaslExecutedFunction){
				out.println(mixElement.name  + "$" + aef.name +
"OUTPUT$" + seg + "[i] * " +

	((AaslExecutedFunction)mixElement).factor[seg] + ";");
			}
		}
		out.println("\t\t" + mix.name  + "$" + aef.name + "OUTPUT$" + seg +"[i]
/= " +
				mix.factorSum[seg] + ";");
		out.println("}");
	}//end print_mix

	void print_outputs(){
		check_outputs();
		for(int i=0; i<segments; i++){
			for(int j=0; j<outputVec.size(); j++){
				AaslMixer2 mix = (AaslMixer2)(outputVec.elementAt(j));
				if(mix.output[i]){
					out.println("double[] " +  aef.name + "OUTPUT$$"
+ i +
							" = new
double[num_samples/segments];");
					out.println("for(int i=0; i<num_samples/segments;
i++){");
```

```
                                        for(int k=0; k<mix.numElements(); k++){
                                                out.print("\t\t" +  aef.name + "OUTPUT$$"
+ i + "[i] += ");
                                                AaslType mixElement =
(AaslType)mix.v.elementAt(k);
                                                if(mixElement instanceof AaslOscillator2){
                                                        out.println(mixElement.name  + "$"
+ aef.name + i + ".output[i] * " +

        ((AaslOscillator2)(mixElement)).factor[i] + ";");
                                                }
                                                else if(mixElement instanceof AaslMixer2){
                                                        out.println(mixElement.name  + "$"
+ aef.name + "OUTPUT$" + i + "[i] * " +

        ((AaslMixer2)(mixElement)).factor[i] +";");
                                                }
                                                else if(mixElement instanceof
AaslExecutedFunction){
                                                        out.println(mixElement.name  + "$"
+ aef.name + "OUTPUT$$" + i + "[i] * " +

        ((AaslExecutedFunction)mixElement).factor[i] + ";");
                                                }
                                        }
                                        out.println("\t\t" +  aef.name + "OUTPUT$$" + i
+"[i] /= " +
                                                        mix.factorSum[i] + ";");
                                        out.println("}");
                                }
                        }
                }
        }

        void check_outputs(){
                boolean[] segs = new boolean[segments];
                for(int i=0; i<segments; i++) {segs[i] = false;}
                for(int i=0; i<outputVec.size(); i++){
                        for(int j=0; j<segments; j++){
                                if( ((AaslMixer2)(outputVec.elementAt(i))).output[j] ){
                                        if(!segs[j])
                                                segs[j]=true;
                                        else
                                                throw new AaslException("Bad Output
Assignment.  More " +
                                        "than one mixer assigned in
segment" + j);
                        }
                    }
                }
                for(int i=0; i<segments; i++){
                    if(segs[i] == false)
                        throw new AaslException("Bad Output Assignment.  One
or more" +
                                " segments has no output assigned!");
                }
        }

        int getIndex(AaslType element){
                if(element instanceof AaslOscillator2){
                    for(int i=0; i<oscVec.size(); i++){
                        if( element.name.equals(
((AaslOscillator2)(oscVec.elementAt(i))).name ) )
                            return i;
                    }
                }
                else if(element instanceof AaslMixer2){
                    for(int i=0; i<mixVec.size(); i++){
                        if( element.name.equals(
((AaslMixer2)(mixVec.elementAt(i))).name ) )
                            return i;
                    }
                }
                return -1;
        }
}
```

AaslMain.java

```
/*
 * By Carlos R. Perez.
*/
package aasl;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import aasl.intermediateTypes.AaslException;
import antlr.CommonAST;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.collections.AST;

/**
 * DEBUG option set directly in AaslException
 *
 */
public class AaslMain {

        private static final String BLACK     = "";  //"\033[30m";
        private static final String RED                   = "";  //"\033[31m";
        private static final String GREEN     = "";  //"\033[32m";
        private static final String YELLOW  = "";  //"\033[33m";
        private static final String BLUE       = "";  //"\033[34m";
        private static final String MAGNETA = "";  //"\033[35m";
        private static final String CYAN       = "";  //"\033[36m";
        private static final String WHITE      = "";  // "\033[37m";

        public static void outStatus(String msg) { System.out.println(BLUE+msg);}
        public static void outErr(String msg) {         System.err.println(RED+msg); }
        public static void outInfo(String msg) { System.out.println(GREEN+msg); }

        /**
         * aasl [input_program_file]
         *
         */
        public static void main( String[] args ) {

                /* number of arguments */
                if (args.length > 1 || args.length < 1) {
                        printHowTo();
                        System.exit(-1);
                }

                /* verify input file exists then compile */
                try {
                        InputStream inputFile = (InputStream) new
FileInputStream(args[0]);

                        outStatus("Compiling "+args[0]+"...");
                        String javaOutputFile = compileInputFile(inputFile);

                        /* sanity checks on output file */
```

```
				if (javaOutputFile.matches("\\W"))
				{
					System.err.println("Invalid OUTPUT file name
<"+javaOutputFile+"!");
					System.exit(-1);
				}

				outStatus("Compiling "+javaOutputFile+".java...");
				compileOutputFile(javaOutputFile);

			} catch (FileNotFoundException e) {
				printHeader();
				outErr("File <"+args[0]+"> does not exists!");
				System.exit(-1);
			}
		}

		public static void printHeader() {
			outInfo("Aasl Compiler v0.1");
		}

		public static void printHowTo() {
			printHeader();
			outInfo("aasl [input_file]");
		}

		/**
		 * Calls the lexer,parser,treewalker, and code generator for the inputFile
		 * @param inputFile
		 */
		private static String compileInputFile(InputStream inputFile) {
			String retJavaOutputFile = "";

			AASLLexer lexer = new AASLLexer(inputFile);
			if (lexer.nr_error != 0) {
				outErr("Exited from "+lexer.nr_error+" Lexer Errors!");
				System.exit(-1);
			}
			AASLParser parser = new AASLParser(lexer);
			AaslAntlrWalker treeWalker = new AaslAntlrWalker();
			CodeGenerator cg;

			try {

				parser.program();
				if (parser.nr_error != 0) {
					outErr("Exited from "+parser.nr_error+" Parser Errors!");
					System.exit(-1);
				}
				AST rootTree = parser.getAST();
				//outInfo( rootTree.toStringTree() );
				treeWalker.expr(rootTree);

				if (treeWalker.v.size() == 0) {
					throw new AaslException("Bad intermediate
representation! V.size()==0");
				}

				retJavaOutputFile = treeWalker.javaOutputFile;

				/* delete the output file if it exists */
				File javaOutputFile = new File(retJavaOutputFile+".java");
				if (javaOutputFile.exists()) {
					if (!javaOutputFile.delete()) {
						outErr("Before 2nd compile: Error deleting existing
file <"+retJavaOutputFile+">");
						System.exit(-1);
					}
			}
				javaOutputFile = new File(retJavaOutputFile+".wav");
				if (javaOutputFile.exists()) {
					if (!javaOutputFile.delete()) {
						outErr("Before 2nd compile: Error deleting existing
file <"+retJavaOutputFile+">");
						System.exit(-1);
					}
			}

				cg = new CodeGenerator(treeWalker.v);
			} catch( RecognitionException e ) {
	   outErr( "Error: Recognition: " + e );
	} catch( TokenStreamException e ) {
	   outErr( "Error: Token stream: " + e );
	}
	return retJavaOutputFile;
	   }

	   public static void compileOutputFile(String input) {


			File javaOutputFile = new File(input+".java");
			if (!javaOutputFile.exists()) {
```

```
                        outErr("At 2nd compile: Error file <"+javaOutputFile.getName()+
                                        "> doesn't exists!");
                        System.exit(-1);
            }
                javaOutputFile = new File(input+".class");
                if (javaOutputFile.exists()) {
                        if (!javaOutputFile.delete()) {
                                outErr("At 2nd compile: Error deleting existing file
<"+javaOutputFile.getName()+">");
                                System.exit(-1);
                        }
            }


                /*
                 * COMPILE JAVA OUTPUT FILE
                 */
                String command = "";
                try {
              // Execute a command with an argument
              command = "javac "+input+".java";
              System.out.println(command);
              Process child = Runtime.getRuntime().exec(command);

              child.waitFor();
              if (child.exitValue() != 0) {
                  System.err.println("Error running <"+command+">");
                  System.exit(-1);
              }

              /*
                     * RUN COMPILED JAVA OUTPUT FILE
                     */
              command = "java "+input;
              System.out.println(command);
              child = Runtime.getRuntime().exec(command);

              child.waitFor();
              if (child.exitValue() != 0) {
                  outErr("Error running <"+command+">");
                  System.exit(-1);
              }
            } catch (IOException e) {
                outErr("Error running <"+command+">");
                System.exit(-1);
            } catch (InterruptedException e) {
```

```
                        e.printStackTrace();
                }
        }
}
```

Test Program

AaslLexerTester.java

```
/*
 * AaslLexerTester.java
 * Carlos R. Perez,
 * Albert Tsai
*/
package aasl.tests;

/*
 * LexerTestCase is a subclass of JUnit's TestCase,
 * so all of the methods you're used to are still available.
 */
import java.io.StringReader;
import org.norecess.antlr.LexerTestCase;
import antlr.*;
import aasl.*;

public class AaslLexerTester extends LexerTestCase implements
AASLParserTokenTypes {
        //@Override
        protected TokenStream makeLexer(String input) {
                return new AASLLexer(new StringReader(input));
        }

        /*
         *
============================================================
====
         *                                  LEXICAL SCANNER TESTS
         *
============================================================
====
         */
        /**
         * @param The first argument is the expected type of the token
         * as specified in the lexer.
         * @param The second argument is the EXPECTED TEXT of the token.
         * @param The third argument is the text to lex (lexify? scan!).
         *
 * @throws TokenStreamException
 */
public void testPlus() throws TokenStreamException {
	assertToken(PLUS, "+", "\n\r  +   \r\n\r");
}
public void testMinus() throws TokenStreamException {
	assertToken(MINUS, "-", "\n\r  -   \r\n\r");
}
public void testTimes() throws TokenStreamException {
	assertToken(TIMES, "*", "\n\r  *   \r\n\r");
}
public void testDiv() throws TokenStreamException {
	assertToken(DIV, "/", "\n\r  /   \r\n\r");
}
public void testMod() throws TokenStreamException {
	assertToken(MOD, "%", "\n\r  %   \r\n\r");
}
public void testAssign() throws TokenStreamException {
	assertToken(ASSIGN, "=", "\n\r  =   \r\n\r");
}
public void testNot() throws TokenStreamException {
	assertToken(NOT, "!", "\n\r  !   \r\n\r");
}
public void testLT() throws TokenStreamException {
	assertToken(LT, "<", "\n\r  <   \r\n\r");
}
public void testGT() throws TokenStreamException {
	assertToken(GT, ">", "\n\r  >   \r\n\r");
}
public void testLE() throws TokenStreamException {
	assertToken(LE, "<=", "\n\r  <=   \r\n\r");
}
public void testGE() throws TokenStreamException {
	assertToken(GE, ">=", "\n\r  >=   \r\n\r");
}
public void testEQ() throws TokenStreamException {
	assertToken(EQ, "==", "\n\r  ==   \r\n\r");
}
public void testNE() throws TokenStreamException {
	assertToken(NE, "!=", "\n\r  !=   \r\n\r");
}
public void testAnd() throws TokenStreamException {
	assertToken(AND, "&&", "\n\r  &&   \r\n\r");
}
public void testOr() throws TokenStreamException {
	assertToken(OR, "||", "\n\r  ||   \r\n\r");
```

```
}
public void testInc() throws TokenStreamException {
        assertToken(INC, "++", "\n\r  ++   \r\n\r");
}
public void testDec() throws TokenStreamException {
        assertToken(DEC, "--", "\n\r  --   \r\n\r");
}
public void testComma() throws TokenStreamException {
        assertToken(COMMA, ",", "\n\r  ,   \r\n\r");
}
public void testSemic() throws TokenStreamException {
        assertToken(SEMIC, ";", "\n\r  ;   \r\n\r");
}
public void testDQuote() throws TokenStreamException {
        assertToken(DQUOTE, "\"", "\n\r  \"   \r\n\r");
}
public void testDQuote2() throws TokenStreamException {
        String dquote = "\"";
        //The whitespace is not liked.
        assertToken(DQUOTE, dquote, "\n\r   " + dquote + "  \r\n\r");
}
public void testLParen() throws TokenStreamException {
        assertToken(LPAREN, "(", "\n\r  (   \r\n\r");
}
public void testRParen() throws TokenStreamException {
        assertToken(RPAREN, ")", "\n\r  )   \r\n\r");
}
public void testLBrace() throws TokenStreamException {
        assertToken(LBRACE, "{", "\n\r  {   \r\n\r");
}
public void testRBrace() throws TokenStreamException {
        assertToken(RBRACE, "}", "\n\r  }   \r\n\r");
}
public void testLBrack() throws TokenStreamException {
        assertToken(LBRACK, "[", "\n\r  [   \r\n\r");
}
public void testRBRACK() throws TokenStreamException {
        assertToken(RBRACK, "]", "\n\r  ]   \r\n\r");
}

public void testInt() throws TokenStreamException {
  assertToken(INT, "123", "123");
  assertToken(INT, "123", " 123 ");
  assertToken(INT, "123", "123 ");
}
public void testFloat() throws TokenStreamException {
		assertToken(FLOAT, "3.", "\n  3.   \n");
	}
	public void testID() throws TokenStreamException {
		assertToken(ID, "testOne", "\n\r  testOne   \r\n\r");
		assertToken(OUTPUT, "OUTPUT", " OUTPUT "); //KEYWORD
		assertToken(ASSIGN,"=", " = ");
		assertToken(STRING, "test.wav","  \"test.wav\"  ");
	}
	public void testLiterals() throws TokenStreamException {
		//Or could be called, "testKeywords"
		//Not "LITERALS_<token name>"?
		//Literals are case-sensitive
		assertToken(START, "start", "start");
		assertToken(END, "end", "end");
		assertToken(FUNC, "func", "func");
		assertToken(MAIN, "main", "main");
		assertToken(DEF, "def", "def");
		assertToken(CONNECT, "connect", "connect");
		assertToken(OUTPUT, "OUTPUT", "OUTPUT");
		assertToken(TONELENGTH, "TONELENGTH", "TONELENGTH");
		assertToken(SEGMENTS, "SEGMENTS", "SEGMENTS");
		assertToken(SEG, "SEG", "SEG");
	}
	public void testNumber() throws TokenStreamException {
		//assertToken(NUMBER, "12345", "\n\r  12345   \r\n\r");
		//assertToken(NUMBER, "12345.", "\n\r  12345. \r\n\r");
	}
	public void testString() throws TokenStreamException {
		assertToken(STRING, "welcome", "\n\r\n\r  \"welcome\"  \r\n");
	}
	public void testComments() throws TokenStreamException {
		assertToken(ID, "ident2", "ident2 // welcome");
		assertToken(OUTPUT, "OUTPUT", "OUTPUT /*testComment
\n\n\n\r\n*/");
	}
}
```

AaslParserTester.java

```
/*
 * AaslParserTester.java
 * Carlos R. Perez,
 * Albert Tsai
*/
package aasl.tests;
```

```
import java.io.StringReader;

import org.norecess.antlr.ParserTestCase;
import org.norecess.antlr.TestableParser;
import aasl.*;
import antlr.*;
import antlr.collections.AST;

public class AaslParserTester extends ParserTestCase implements
AASLParserTokenTypes {
	//@Override
	protected TokenStream makeLexer(String input) {
		AASLLexer ret = new AASLLexer(new StringReader(input));
		return ret;
	}
	protected TestableParser makeParser(TokenStream lexer) {
		AASLParser parser = new AASLParser(lexer);

	parser.getASTFactory().setASTNodeClass("org.norecess.antlr.LineNumberAST")
;
		return parser;
	}


	/*
	 *
================================================================
====
	 *						PARSER AST TESTS
	 *
================================================================
====
	 * http://cs.calvin.edu/curriculum/cs/382/jdfrenscompilers/parser.html
	 * more info @ http://antlr-testing.sourceforge.net/
	 *
	 * assertPreroder(message, expectedTokenType, expectedPreorderstring,
producedAST);
    *
    * MESSAGE: is a standard JUnit message; describe the test.
    *
    * EXPECTED TOKEN TYPE: is the expected token type of the
    * root of the produced AST (tokens from the lexer and parser).
    *
    * EXPECTED PREORDER STRING: is a String representation of a fully
    * parenthesized preorder traversal of the expected AST.
```

```
*
* PRODUCED AST: is the expression to be tested; helper methods make
* this easy to write.
       */
      /* *********************************
       * 4 Expressions
       * *********************************/

      public void testAssignment() {
              System.out.println("testAssignment():");
              AST testAST = produce("assignment", "test123t_45 =  69");
              System.out.println(" =>"+testAST.toStringTree());
              assertPreorder("parse assignment rule",
                              ASSIGN,
                         "(=(test123t_45)(69))",
                         testAST);
      }

      public void testBoolExpr() {

              System.out.println("testBoolExpr():");
              AST testAST = produce("bool", "identifier");
              System.out.println(" =>"+testAST.toStringTree());
              assertPreorder("parse bool rule",
                              ID,
                         "(identifier)",
                         testAST);

              testAST = produce("bool", "0.56565565");
              System.out.println(" =>"+testAST.toStringTree());
              assertPreorder("parse bool rule",
                              FLOAT,
                         "(0.56565565)",
                         testAST);

              testAST = produce("bool", "12382093810923");
              System.out.println(" =>"+testAST.toStringTree());
              assertPreorder("parse bool rule",
                              INT,
                         "(12382093810923)",
                         testAST);

              testAST = produce("bool", "(3*2)");
              System.out.println(" =>"+testAST.toStringTree());
              assertPreorder("parse bool rule",
                              TIMES,
                "(*(3)(2))",
                testAST);

testAST = produce("bool", "-2");
System.out.println(" =>"+testAST.toStringTree());
assertPreorder("parse bool rule",
                        NEGATE,
                "(-(2))",
                testAST);

// TODO: this doesn't make sense semantically!
testAST = produce("bool", "!-2");
System.out.println(" =>"+testAST.toStringTree());
assertPreorder("parse bool rule",
                        NOT,
                "(!(-(2)))",
                testAST);

//
testAST = produce("bool", "!true");
System.out.println(" =>"+testAST.toStringTree());
assertPreorder("parse bool rule",
                        NOT,
                "(!(true))",
                testAST);
testAST = produce("bool", "!identifier");
System.out.println(" =>"+testAST.toStringTree());
assertPreorder("parse bool rule",
                        NOT,
                "(!(identifier))",
                testAST);

testAST = produce("bool", "1 < 2");
System.out.println(" =>"+testAST.toStringTree());
assertPreorder("parse bool rule",
                        LT,
                "(<(1)(2))",
                testAST);
testAST = produce("bool", "((1*1)/1) < 2*1");
System.out.println(" =>"+testAST.toStringTree());
assertPreorder("parse bool rule",
                        LT,
                "(<(/(*(1)(1))(1))(*(2)(1)) )".replaceAll("\\s",""),
                testAST);
testAST = produce("bool", "((1*1)/1) <= 2*1");
System.out.println(" =>"+testAST.toStringTree());
```

```
assertPreorder("parse bool rule",
            LE,
        "(<=(/(*(1)(1))(1))(*(2)(1)) )".replaceAll("\\s",""),
        testAST);
testAST = produce("bool", "((1*1)/1) > 2*1");
System.out.println(" =>"+testAST.toStringTree());
assertPreorder("parse bool rule",
            GT,
        "(>(/(*(1)(1))(1))(*(2)(1)) )".replaceAll("\\s",""),
        testAST);

//
testAST = produce("bool", "((1*1)/1) >= 2*1");
System.out.println(" =>"+testAST.toStringTree());
assertPreorder("parse bool rule",
            GE,
        "(>=(/(*(1)(1))(1))(*(2)(1)) )".replaceAll("\\s",""),
        testAST);

testAST = produce("bool", "((1*1)/1) == 2*1");
System.out.println(" =>"+testAST.toStringTree());
assertPreorder("parse bool rule",
            EQ,
        "(==(/(*(1)(1))(1))(*(2)(1)) )".replaceAll("\\s",""),
        testAST);
testAST = produce("bool", "((1*1)/1) != 2*1");
System.out.println(" =>"+testAST.toStringTree());
assertPreorder("parse bool rule",
            NE,
        "(!=(/(*(1)(1))(1))(*(2)(1)) )".replaceAll("\\s",""),
        testAST);

//
//CURRENTLY A FAILED TEST due to !=
/*
AST testAST = produce("bool", "SEG != 1");
System.out.println(" =>"+testAST.toStringTree());
assertPreorder("parse bool rule",
            SEGSTMT,
        "(SEGSTMT (!=)(1) )".replaceAll("\\s",""),
        testAST);
        */


testAST = produce("bool", "((1*1)/1) && 2*1");
System.out.println(" =>"+testAST.toStringTree());
```

```
                assertPreorder("parse bool rule",
                        AND,
                    "(&&(/(*(1)(1))(1))(*(2)(1)) )".replaceAll("\\s",""),
                    testAST);


                testAST = produce("bool", "((1*1)/1) || 2*1");
                System.out.println(" =>"+testAST.toStringTree());
                assertPreorder("parse bool rule",
                        OR,
                    "(||(/(*(1)(1))(1))(*(2)(1)) )".replaceAll("\\s",""),
                    testAST);

                testAST = produce("bool", "x==1 || SEG==2 || SEG==5");
                System.out.println(" =>"+testAST.toStringTree());
                //Previously tested for
                //"( || ( || ( == (x)(1) ) ( SEGSTMT (==)(2)) ) ( SEGSTMT (==)(5))
)".replaceAll("\\s","")
                assertPreorder("parse bool rule",
                        OR,
                    "( || ( || ( == (x)(1) ) ( SEG (2)) ) ( SEG (5))
)".replaceAll("\\s",""),
                    testAST);

        }


        public void boolSimple() {
                System.out.println("testBoolSimple()");
                AST testAST = produce("join", "x == 1 || SEG == 3 || SEG == 5");
                assertPreorder("Bool Simple",
                                        OR,
                                        "(||(||(==(x)(1))(SEG(3)))(SEG(5)))",
                                        testAST);
        }
        //I am confused about the parent on this one.
        //the parent is always from the top level rule.
        //This is weird how SEG is the root
        //(root(child)(child))
        public void testJoin() {
                System.out.println("testJoin()");
                AST testAST = produce("join", "x == 1 && SEG == 3 && SEG == 5");
                assertPreorder("Join",
                                        AND,
                                        "(&&(&&(==(x)(1))(SEG(3)))(SEG(5)))",
                                        testAST);
```

```
	}

	//the last root is the top
	//odd. But factors are roots too. Explain that.
	public void testEquality() {
		System.out.println("testEquality()");
		AST testAST = produce("equality", "osc1*6+5 >= osc2/10-3 != 55");
		assertPreorder("Equality",
					NE,
					"(!=(>=(+(*(osc1)(6))(5))(-
(/(osc2)(10))(3)))(55))",
					testAST);

		//This is extremely odd, and can be disallowed by the grammar.
		testAST = produce("equality", "10 != 5 == 6");
		assertPreorder("Equality2",
					EQ,
					"(==(!=(10)(5))(6))",
					testAST);

		testAST = produce("equality", "SEG == 10");
		assertPreorder("Equality3",
					SEG,
					"(SEG(10))",
					testAST);

		testAST = produce("equality", "SEG == 10");
		assertPreorder("Equality3",
					SEG,
					"(SEG(10))",
					testAST);
	}

	public void testRel() {
		System.out.println("testRel()");
		AST testAST = produce("rel", "osc1*6+5 >= osc2/10-3");
		assertPreorder("Rel",
					GE,
					"(>=(+(*(osc1)(6))(5))(-(/(osc2)(10))(3)))",
					testAST);
	}


	public void testExpr() {
		System.out.println("testExpr():");
		AST testAST = produce("expr", "1+2*3");
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder("parse addition & substraction rules",
					PLUS,
				"(+(1)(*(2)(3)))",
				testAST);

		testAST = produce("expr", "-1+2*3");
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder("parse addition & substraction rules",
					PLUS,
				"(+(-(1))(*(2)(3)))",
				testAST);

		testAST = produce("expr", "-1+(2*3)/4");
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder("parse addition & substraction rules",
					PLUS,
				"(+(-(1))(/(*(2)(3))(4)))",
				testAST);
	}

	public void testTerm() {
		System.out.println("testTerm()");
		AST testAST = produce("term", "5*10");
		System.out.println(" =>" + testAST.toStringTree());
		assertPreorder("Term",
							TIMES,
							"(*(5)(10))",
							testAST);
		//ok it doesn't like spaces either (as the assert argument)
		//the tree is NOT composed of the tokens, but the actual input.
		//again, the root node is tested, however.
		testAST = produce("term", "5/10");
		System.out.println(" =>" + testAST.toStringTree());
		assertPreorder("Term",
							DIV,
							"(/(5)(10))",
							testAST);
	}

	public void testUnary() {
		System.out.println("testUnary()");
		AST testAST = produce("unary", "-10");
		System.out.println(" =>" + testAST.toStringTree());
		assertPreorder("Unary",
							NEGATE,
                                        "(-(10))",
                                        testAST);
          //do the parentheses matter or not??

          //what the hell is this?
          testAST = produce("unary", "!10");
          System.out.println(" =>" + testAST.toStringTree());
          assertPreorder("Unary",
                                        NOT,
                                        "(!(10))",
                                        testAST);

          testAST = produce("unary", "10");
          System.out.println(" =>" + testAST.toStringTree());
          assertPreorder("Unary",
                                        INT,
                                        "(10)",
                                        testAST);
}

public void testFactor() {
          System.out.println("testFactor()");
          AST testAST = produce("factor", "5");
          assertPreorder("Factor (int)",
                                        INT,
                                        "(5)",
                                        testAST);

          testAST = produce("factor", "5.0");
          assertPreorder("Factor (float)",
                                        FLOAT,
                                        "(5.0)",
                                        testAST);
          //odd, fails without the (), but toStringTree doesn't show it.

          System.out.println(" =>" + testAST.toStringTree());
          testAST = produce("factor", "albert");
          assertPreorder("Factor (ID)",
                              ID,
                              "(albert)",
                              testAST);
          System.out.println(" =>" + testAST.toStringTree());

          //test bool rule later
}
```

```
	/* **********************************
	 * 5 Definitions
	 * *********************************/
	//defs?

	/* **********************************
	 * 5.2 Oscillator Definition
	 * *********************************/

	public void testOscDef1() {
		AST testAST = produce("oscdef", "osc m2342342o=  \t\n\"SINE\" 
//\t\ttestComment");
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder("parse oscdef rule",
				OSCDEF,
			"(OSCDEF(m2342342o)(SINE))",
			testAST);
	}
	public void testOscDef2() {
		AST testAST = produce("oscdef", "osc m2342342o=  \t\n\"SQUARE\" 
//\t\ttestComment");
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder("parse oscdef rule",
				OSCDEF,
			"(OSCDEF(m2342342o)(SQUARE))",
			testAST);
	}
	public void testOscDef3() {
		AST testAST = produce("oscdef", "osc m2342342o=  \t\n\"SAW\" 
//\t\ttestComment");
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder("parse oscdef rule",
				OSCDEF,
			"(OSCDEF(m2342342o)(SAW))",
			testAST);
	}
	public void testOscDef4() {
		AST testAST = produce("oscdef", "osc m2342342o=  \t\n\"REVSAW\" 
//\t\ttestComment");
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder("parse oscdef rule",
				OSCDEF,
			"(OSCDEF(m2342342o)(REVSAW))",
			testAST);
	}
	public void testOscDef5() {
```

```
            //Notice the rule (defs instead of oscdef) this is using!
            AST testAST = produce("defs", "osc o1 = \"SINE\";");
            System.out.println(" =>"+testAST.toStringTree());
            String expected = "" +
            "(OSCDEF" +
    "       (o1)" +
    "       (SINE))";
            expected = expected.replaceAll("\\s","");
            assertPreorder("parse oscdef rule",
                            OSCDEF,
                            expected,
                    testAST);
    }

    /* *********************************
     * 5.3 Oscillator Bank Definition
     * *********************************/

    public void testOscbDef() {
            AST testAST = produce("oscbdef", "oscbank ob1 = 3");
            System.out.println(" =>"+testAST.toStringTree());
            assertPreorder("parse oscbdef rule",
                            OSCBANKDEF,
                    "(OSCBANKDEF(ob1)(3))",
                    testAST);
    }

    public void testOscbElemDef() {
            AST testAST = produce("oscbelementdef", "ob1[3] = \"SINE\"");
            System.out.println(" =>"+testAST.toStringTree());
            assertPreorder("parse oscbelementdef rule",
                            OSCBANKELEMENT_DEF,
                    "(OSCBANKELEMENT_DEF(ob1)(3)(SINE))",
                    testAST);
    }

    /* *********************************
     * 5.4 Envelope Definition
     * *********************************/

    public void testEnvDef() {
            AST testAST = produce("envdef", "env envv = { (0.0,0.3) (0.2,0.2)
(0.4,0.2) (0.8,0.8) (1.0,0.) }");
            System.out.println(" =>"+testAST.toStringTree());
            assertPreorder("parse envdef rule",
                            ENVDEF,
"(ENVDEF(envv)(0.0)(0.3)(0.2)(0.2)(0.4)(0.2)(0.8)(0.8)(1.0)(0.))",
                    testAST);
    }

    /* *********************************
     * 5.5 Mixer Definition
     * *********************************/
    /* How to assert bad/failure?
    public void testMixDef1() {
            System.out.println("testDefs1():");
            AST testAST = produce("mixdef", "mixer m1 =4");
            System.out.println(" =>"+testAST.toStringTree());
            assertPreorder("parse mixdef rule",
                            MIXERDEF,
                    "(MIXERDEF(m1)(4))",
                    testAST);
    }*/

    public void testMixDef2() {
            System.out.println("testDefs2():");
            AST testAST = produce("mixdef", "mixer m1");
            System.out.println(" =>"+testAST.toStringTree());
            assertPreorder("parse mixdef rule",
                            MIXERDEF,
                    "(MIXERDEF(m1))",
                    testAST);
    }

    /* *********************************
     * 6 Statements
     * *********************************/
    /*
     * conditionalDef?
     * conditionalCon?
     * ifCon?
     * forCon?
     */

    /* *********************************
     * 6.1 Conditionals
     * *********************************/
    //conditionaldef

    /* *********************************
     * 6.1.1 For loops
```

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

	//[DEFINITION SECTION]
	public void testForDef1() {
		System.out.println("testForDef1():");
		String test = ""+
		"for(x=1;x<3;x++) \n"+
		"{ \n"+
		"	ob1[x] = \"SQUARE\"; \n"+
		"} \n";
		AST testAST = produce("fordef", test);
		System.out.println(" =>"+testAST.toStringTree());
		String expected = ""+
		"(for "+
		"		(=(x)(1)) "+
		"		(<(x)(3)) "+
		"		(++(x)) "+
		"		(DEFSTATEMENTS "+
		"
	(OSCBANKELEMENT_DEF(ob1)(x)(SQUARE))) "+
		")";
		expected = expected.replaceAll("\\s","");
		assertPreorder("ForDef",
				FOR,
			 expected,
			 testAST);
	}

	//[CONNECTION SECTION]
	public void testForCon() {
		System.out.println("testForCon()");
		String test = "" +
		"for(x=1; x<3; x++) //for loop featuring .size feature (returns # of osc in
\n" +
		"{ //oscbank \n" +
		"	ob1[x](440, 0.5); //apostrophes differentiate variables \n" +
		"	ob1[x](220*x, ob1[x-1]); \n" +
		"} ";
		AST testAST = produce("forcon", test);
		String expected = ""+
		"(for "+
		"		(=(x)(1)) "+
		"		(<(x)(3)) "+
		"		(++(x)) "+
		"		(CONSTATEMENTS "+
                "
        (OSCCON(ob1)(BANKINDEX(x))(OSCCONARG1(440)(BANKINDEX))(OSC
CONARG2(0.5)(BANKINDEX))) "+
                "
        (OSCCON(ob1)(BANKINDEX(x))(OSCCONARG1(*(220)(x))(BANKINDEX))
(OSCCONARG2(ob1)(BANKINDEX(-(x)(1))))) "+
                "               ) "+
                ") ";
                expected = expected.replaceAll("\\s","");
                System.out.println(" =>"+testAST.toStringTree());
                assertPreorder("Fordef",
                                FOR,
                        expected,
                        testAST);
        }

        /* ********************************
         * 6.1.2 If/Else statements
         * ********************************/

        //[DEFINITION SECTION]
        //can you have an If without an Else? (grammar should fail)
        public void testIfdef() {
                System.out.println("testIfdef():");
                String test = "" +
                //slightly crippled example (not SEG % 2)
                //If you don't have "\n" at the end of the strings,
                //then the comments will continue to the end of the test string.
                //This leads to failure.
                "if(SEG == 0) //set waveform of oscbank elements depending on which\n
" +
                "{                              //time segment you're in\n " +
                "       ob1[1] = \"SQUARE\"; " +
                "       ob1[2] = \"SINE\";  " +
                "} " +
                "else " +
                "{ " +
                "       ob1[1] = \"SINE\"; " +
                "       ob1[2] = \"SAW\"; " +
                "}";
                AST testAST = produce("ifdef", test);
                System.out.println(" =>" + testAST.toStringTree());
                //Odd, now the node names are in the expected tree (unlike expression
parse trees)
                //Well in the expressions, the AND's were tokens. If you resolve to a
token, use token value (the literal)
```

```
		//for the name
		//if you resolve to a rule, use that for the name.
		//Nice it doesn't also ask for the intermediate rules. Only the resolved
rule/token
		String expected = "" +
		"(if " +
		"		(SEG(0)) " +
		"		(DEFSTATEMENTS " +
		"			(OSCBANKELEMENT_DEF "+
		"				(ob1) "+
		"				(1) "+
		"				(SQUARE)) "+
		"			(OSCBANKELEMENT_DEF "+
		"				(ob1) "+
		"				(2) "+
		"				(SINE)) " +
		"		) " +
		"		(DEFSTATEMENTS " +
		"
	(OSCBANKELEMENT_DEF(ob1)(1)(SINE))(OSCBANKELEMENT_DEF(ob1
)(2)(SAW)) " +
		"		) " +
		"	) ";
		expected = expected.replaceAll("\\s","");
		assertPreorder("ifdef",
				IF,
				expected,
				testAST);
	}

	//[CONNECTION SECTION]
	public void testIfcon() {
		System.out.println("testIfcon():");
		String test = "" +
		"if(x==1 || SEG==2 || SEG==5) " +
		"{ " +
		"	ob1[x](440, 0.5); //apostrophes differentiate variables \n" +
		"} " +
		"else " +
		"{ " +
		"	ob1[x](220*x, ob1[x-1]); " +
		"} ";
		AST testAST = produce("ifcon", test);
		System.out.println(" =>" + testAST.toStringTree());
		String expected = "" +
		"(if "+
        "               (|| "+
        "                               (|| "+
        "                                               (== "+
        "                                                       (x) "+
        "                                                       (1)) "+
        "                                               (SEG(2)) "+
        "                               ) "+
        "                               (SEG(5)) "+
        "               ) "+
        "               (CONSTATEMENTS "+
        "                               (OSCCON "+
        "                                               (ob1) "+
        "                                               (BANKINDEX(x)) "+
        "                                               (OSCCONARG1 "+
        "                                                               (440) "+
        "
(BANKINDEX)) "+
        "                                               (OSCCONARG2 "+
        "                                                               (0.5) "+
        "
(BANKINDEX)) "+
        "                               ) "+
        "               ) "+
        "               (CONSTATEMENTS "+
        "                               (OSCCON "+
        "                                               (ob1) "+
        "                                               (BANKINDEX(x)) "+
        "                                               (OSCCONARG1 "+
        "                                                               (* "+
        "
(220) "+
        "
(x)) "+
        "
(BANKINDEX)) "+
        "                                               (OSCCONARG2 "+
        "                                                               (ob1) "+
        "
(BANKINDEX "+
        "
(- "+
        "
        (x) "+
        "
        (1)) "+
        "                                                               ) "+
            "                                                   ) "+
            "                                  ) "+
            "                 ) "+
            ") ";
            expected = expected.replaceAll("\\s","");
            assertPreorder("ifdef",
                           IF,
                           expected,
                           testAST);
    }

    public void testDelta() {
            System.out.println("testDelta():");
            AST testAST = produce("delta", "test123t_45--");
            System.out.println(" =>"+testAST.toStringTree());
            assertPreorder("parse delta rule",
                           DEC,
                       "(--(test123t_45))",
                       testAST);

            testAST = produce("delta", "test123t_45++");
            System.out.println(" =>"+testAST.toStringTree());
            assertPreorder("parse delta rule",
                           INC,
                       "(++(test123t_45))",
                       testAST);
    }

    /* *********************************
     * 6.2 Connection Statements
     * *********************************/
    //constatements



    /* *********************************
     * 6.2.1 Oscillator Connections
     * *********************************/

    public void testOscCons1() {
            System.out.println("testOscCons1():");
            //EOF crap?
            //BANKINDEX is popping up in the weirdest places
            AST testAST = produce("osccon", "o1(440, e1); //apostrophes
differentiate variables");
            System.out.println(" =>"+testAST.toStringTree());
```

```
                String expected = "" +
                        "(OSCCON " +
                        "         (o1)(BANKINDEX)" +
                        "         (OSCCONARG1(440)(BANKINDEX))" +
                        "         (OSCCONARG2(e1)(BANKINDEX))" +
                        ")";
                expected = expected.replaceAll("\\s","");
                assertPreorder("parse osccons rule",
                                OSCCON,
                           expected,
                           testAST);
        }

        public void testOscCons2() {
                System.out.println("testOscCons2():");
                AST testAST = produce("osccon", "o2(e2@1000, ob1[2]); //apostrophes
differentiate variables");
                System.out.println(" =>"+testAST.toStringTree());
                String expected = ""+
                "(OSCCON "+
                "               (o2)(BANKINDEX) "+
                "               (OSCCONARG1 "+
                "                       (e2)(BANKINDEX)(1000) "+
                "               ) "+
                "               (OSCCONARG2 "+
                "                       (ob1) "+
                "                       (BANKINDEX(2)) "+
                "               ) "+
                ") ";
                expected = expected.replaceAll("\\s","");
                assertPreorder("parse osccons rule",
                                OSCCON,
                           expected,
                           testAST);
        }

        public void testOscCons3() {
                System.out.println("testOscCons3():");
                AST testAST = produce("osccon", "ob1[x](440, 0.5); //apostrophes
differentiate variables");
                System.out.println(" =>"+testAST.toStringTree());
                String expected = ""+
                "(OSCCON "+
                "               (ob1) "+
                "               (BANKINDEX(x)) "+
                "               (OSCCONARG1(440)(BANKINDEX)) "+
                "           (OSCCONARG2(0.5)(BANKINDEX)) "+
                ")";
                expected = expected.replaceAll("\\s","");
                assertPreorder("parse osccons rule",
                                OSCCON,
                           expected,
                           testAST);
        }

        public void testOscCons4() {
                System.out.println("testOscCons():");
                AST testAST = produce("osccon", "ob1[x](220*x, ob1[x-1]);");
                System.out.println(" =>"+testAST.toStringTree());
                String expected = ""+
                "(OSCCON "+
                "           (ob1) "+
                "           (BANKINDEX(x)) "+
                "           (OSCCONARG1 "+
                "                   (* "+
                "                   (220) "+
                "                   (x))" +
                "                   (BANKINDEX) "+
                "           ) "+
                "           (OSCCONARG2 "+
                "                   (ob1) "+
                "                   (BANKINDEX "+
                "                       (- "+
                "                           (x) "+
                "                           (1))) "+
                "           ) "+
                ")";
                expected = expected.replaceAll("\\s","");
                assertPreorder("parse osccons rule",
                                OSCCON,
                           expected,
                           testAST);
        }

        public void testBankIndex1() {
                System.out.println("testBankIndex1():");
                AST testAST = produce("bankIndex", "[5]");
                System.out.println(" =>"+testAST.toStringTree());
                assertPreorder("bankIndex",
                                BANKINDEX,
                           "(BANKINDEX(5))",
                           testAST);
```

```
	}

	public void testBankIndex2() {
		System.out.println("testBankIndex2():");
		AST testAST = produce("bankIndex", "[x-1]");
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder("bankIndex2",
				BANKINDEX,
			"(BANKINDEX(-(x)(1)))",
			testAST);
	}
	public void testOscconcarg1() {
		System.out.println("testOscconarg1():");
		AST testAST = produce("oscconargs1", "440");
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder("oscconarg",
				OSCCONARG1,
			"(OSCCONARG1(440)(BANKINDEX))",
			testAST);
	}

	public void testOscconcarg2() {
		System.out.println("testOscconarg2():");
		AST testAST = produce("oscconargs1", "0.5");
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder("oscconarg",
				OSCCONARG1,
			"(OSCCONARG1(0.5)(BANKINDEX))",
			testAST);
	}

	public void testOscconcarg3() {
		System.out.println("testOscconarg3():");
		AST testAST = produce("oscconargs1", "200*x");
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder("oscconarg",
				OSCCONARG1,
			"(OSCCONARG1(*(200)(x))(BANKINDEX))",
			testAST);
	}

	public void testOscconcarg4() {
		System.out.println("testOscconarg4():");
		AST testAST = produce("oscconargs1", "ob1[x-1]");
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder("oscconarg",
                        OSCCONARG1,
                    "(OSCCONARG1(ob1)(BANKINDEX(-(x)(1))))",
                    testAST);
        }

        public void testOscconcarg5() {
                System.out.println("testOscconarg5():");
                AST testAST = produce("oscconargs1", "ob1[x]@500.0");
                System.out.println(" =>"+testAST.toStringTree());
                assertPreorder("oscconarg",
                        OSCCONARG1,
                    "(OSCCONARG1(ob1)(BANKINDEX(x))(500.0))",
                    testAST);
        }

        /* **********************************
         * 6.2.2 Mixer Connections
         * *********************************/

        public void testMixStmt() {
                System.out.println("testMixStmt():");
                AST testAST = produce("mixstmt", "test123t_45=4;");
                System.out.println(" =>"+testAST.toStringTree());
                assertPreorder("parse mix stmt rule",
                        MIXSTMT,
                    "(MIXSTMT(test123t_45)(4))",
                    testAST);

                testAST = produce("mixstmt", "test123t_45 = id2 * 3;");
                System.out.println(" =>"+testAST.toStringTree());
                assertPreorder("parse mix stmt rule",
                        MIXSTMT,
                    "(MIXSTMT(test123t_45)(*(id2)(3)))",
                    testAST);

                testAST = produce("mixstmt", "test123t_45 = id2{2.002} * 3;");
                System.out.println(" =>"+testAST.toStringTree());
                assertPreorder("parse mix stmt rule",
                        MIXSTMT,

"(MIXSTMT(test123t_45)(*(FUNCCALL(id2)(EXPR_LIST(2.002)))(3)))",
                    testAST);

                testAST = produce("mixstmt", "test123t_45 = 3.0008*id2{2.002}\n;");
                System.out.println(" =>"+testAST.toStringTree());
                assertPreorder("parse mix stmt rule",
                                MIXSTMT,

"(MIXSTMT(test123t_45)(*(3.0008)(FUNCCALL(id2)(EXPR_LIST(2.002)))))",
                                testAST);

                testAST = produce("mixstmt", "S1 = 0.4*o1 + 0.6*o2;");
                System.out.println(" =>"+testAST.toStringTree());
                assertPreorder("parse mix stmt rule",
                                MIXSTMT,
                        "(MIXSTMT(S1) (+ (*(0.4)(o1)) (*(0.6)(o2)))
)".replaceAll("\\s",""),
                        testAST);
        }

        public void testMixcon() {
                System.out.println("testMixterms()");
                AST testAST = produce("mixcon", "5 * osc1 + 10 * osc2");
                System.out.println(" =>"+testAST.toStringTree());
                assertPreorder("Mixcon",
                                PLUS,
                        "(+(*(5)(osc1))(*(10)(osc2)))",
                        testAST);
        }

        public void testMixterms() {
                System.out.println("testMixterms()");
                AST testAST = produce("mixterms", "5 * osc1 + 10 * osc2");
                System.out.println(" =>"+testAST.toStringTree());
                assertPreorder("Mixterm",
                                PLUS,
                        "(+(*(5)(osc1))(*(10)(osc2)))",
                        testAST);
        }

        //you will get a "parse error" if you feed more input than
        //the rule can handle. You will also get a parse error if you
        //feed "5 5", but tell assert to expect one INT.
        public void testMixterm() {
                System.out.println("testMixterm()");
                AST testAST = produce("mixterm", "5");
                System.out.println(" =>"+testAST.toStringTree());
                assertPreorder("Mixterm",
                                INT,
                        "(5)",
                        testAST);
```

```
		testAST = produce("mixterm", "5 * osc1");
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder("Mixterm",
				TIMES,
			"(*(5)(osc1))",
			testAST);
	}

	/* *********************************
	 * 7 Functions
	 * *********************************/

	public void testFuncDef() {
		System.out.println("testFuncDef():");
		String test = ""+
		"start func o2o (int x, float y)"+
		"	start def"+
		"		osc o1=\"SAW\";"+
		"		osc o2=\"SINE\";"+
		"	end def"+
		"	start connect"+
		"		o1(x, o2);"+
		"		o2(0.5, 0.75);"+
		"	end connect"+
		"	OUTPUT = o1;"+
		"end func o2o";
		String expected = "" +
		"(FUNCDEF "+
		"		(o2o) "+
		"		(ARG_LIST_DECLARATION(int)(x)(float)(y)) "+
		"
	(DEFSTATEMENTS(OSCDEF(o1)(SAW))(OSCDEF(o2)(SINE))) "+
		"
	(CONSTATEMENTS(OSCCON(o1)(BANKINDEX)(OSCCONARG1(x)(BANK
INDEX))(OSCCONARG2(o2)(BANKINDEX)))(OSCCON(o2)(BANKINDEX)(OSCC
ONARG1(0.5)(BANKINDEX))(OSCCONARG2(0.75)(BANKINDEX)))) "+
		"		(OUTPUT(o1)))";
		expected = expected.replaceAll("\\s","");
		AST testAST = produce("funcdef", test);
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder("parse funcdef rule",
				FUNCDEF,
			expected,
			testAST);
	}
```

```
//arg_list_decl?
//data_type?

public void testArgList() {
        System.out.println("testArgList():");
        AST testAST = produce("arg_list", "123, 123.4, a, asd");
        System.out.println(" =>"+testAST.toStringTree());
          assertPreorder("parse arg_list rule",
                    ARG_LIST,
            "(ARG_LIST(123)(123.4)(a)(asd))",
            produce("arg_list", "123, 123.4, a, asd"));
        }

//expr_list?
//funccall?

public void testExprList() {
        System.out.println("testExprList():");
        AST testAST = produce("expr_list", "test123t_45");
        System.out.println(" =>"+testAST.toStringTree());
        assertPreorder("parse expr_list rule",
                        EXPR_LIST,
                "(EXPR_LIST(test123t_45))",
                testAST);

        testAST = produce("expr_list", "test123t_45, id2, id3");
        System.out.println(" =>"+testAST.toStringTree());
        assertPreorder("parse expr_list rule",
                        EXPR_LIST,
                "(EXPR_LIST(test123t_45)(id2)(id3))",
                testAST);

        testAST = produce("expr_list", "-4, x");
        System.out.println(" =>"+testAST.toStringTree());
        assertPreorder("parse expr_list rule",
                        EXPR_LIST,
                "(EXPR_LIST(-(4))(x))",
                testAST);
}

/* *********************************
 * 7.1 Definition Section
 * *********************************/

public void testDefSection() {
```

```
                System.out.println("testDefSection():");
                AST testAST = produce("defsec",
"       start def"+
"               osc o1=\"SAW\";"+
"               osc o2=\"SINE\";"+
"       end def");
                System.out.println(" =>"+testAST.toStringTree());
                String expected = ""+
            "(DEFSTATEMENTS " +
            "    (OSCDEF" +
            "            (o1)" +
            "            (SAW))" +
            "    (OSCDEF" +
            "            (o2)" +
            "            (SINE))" +
            ")";
                expected = expected.replaceAll("\\s","");
                assertPreorder("parse defsec rule",
                                DEFSTATEMENTS,
                                expected,
                        testAST);

                String test = ""+
                "start def "+
                "       oscbank oscb1 = 2; "+
                "       oscb1[0]=\"SINE\"; "+
                "       oscb1[1]=\"SQUARE\"; "+
                "       mixer s1; "+
                "end def ";
                testAST = produce("defsec", test);
                expected = " "+
                "(DEFSTATEMENTS "+
                "               (OSCBANKDEF "+
                "                               (oscb1) "+
                "                               (2)) "+
                "               (OSCBANKELEMENT_DEF "+
                "                               (oscb1) "+
                "                               (0) "+
                "                               (SINE)) "+
                "               (OSCBANKELEMENT_DEF "+
                "                               (oscb1) "+
                "                               (1) "+
                "                               (SQUARE)) "+
                "               (MIXERDEF "+
                "                               (s1)) "+
                ") ";
                expected = expected.replaceAll("\\s","");
                assertPreorder("Definition Section",
                                DEFSTATEMENTS,
                                expected,
                        testAST);
        }

        //defstatements?

        /* ********************************
         * 7.2 Connection Section
         * ********************************/

        public void testConnectSection() {
                String test = ""+
                "start connect "+
                "       oscb1[0](0.5,1.0); "+
                "       oscb1[1](440,oscb1[0]); "+
                "       s1 = oscb1[1]; "+
                "end connect";
                AST testAST = produce("consec", test);
                String expected = " "+
                "(CONSTATEMENTS "+
                "           (OSCCON "+
                "                       (oscb1) "+
                "                       (BANKINDEX(0)) "+
                "                       (OSCCONARG1(0.5)(BANKINDEX)) "+
                "                       (OSCCONARG2(1.0)(BANKINDEX)) "+
                "           ) "+
                "           (OSCCON "+
                "                       (oscb1) "+
                "                       (BANKINDEX(1)) "+
                "                       (OSCCONARG1(440)(BANKINDEX)) "+
                "
        (OSCCONARG2(oscb1)(BANKINDEX(0))) "+
                "           ) "+
                "           (MIXSTMT "+
                "                       (s1) "+
                "                       (OSCBANK "+
                "                                   (oscb1) "+
                "                                   (1)) "+
                "           ) "+
                ") ";
                expected = expected.replaceAll("\\s","");
                assertPreorder("Definition Section",
                                CONSTATEMENTS,
                                expected,
                        testAST);

        }

        /* *********************************
         * 7.3 Return Value
         * *********************************/

        public void testOutputeq() {
                String test = "OUTPUT = s1;";
                AST testAST = produce("outputeq", test);
                String expected = "(OUTPUT(s1))";
                expected = expected.replaceAll("\\s","");
                assertPreorder("Return Value",
                                OUTPUT,
                                expected,
                        testAST);

        }

        /* *********************************
         * 9 AASL Program
         * *********************************/

        public void testProgram() {
                String test = ""+
                "start header \n"+
                "OUTPUT = \"secondtest\" \n"+
                "TONELENGTH = 3 \n"+
                "end header \n"+
                " \n"+
                "start func main \n"+
                "       start def \n"+
                "               //osc o1 = \"SINE\" \n"+
                "               //osc o2 = \"SQUARE\" \n"+
                "               oscbank oscb1 = 2; \n"+
                "               oscb1[0]=\"SINE\"; \n"+
                "               oscb1[1]=\"SQUARE\"; \n"+
                "               mixer s1; \n"+
                "       end def \n"+
                "        \n"+
                "       start connect \n"+
                "               oscb1[0](0.5,1.0); \n"+
                "               oscb1[1](440,oscb1[0]); \n"+
                "               s1 = oscb1[1]; \n"+
                "       end connect \n"+
                "        \n"+
                "       OUTPUT = s1; \n"+
                "end func main \n";
                AST testAST = produce("program", test);
                String expected = ""+
                "(PROGRAM "+
                "               (HEAD "+
                "                               (OUTPUT(secondtest)) "+
                "                               (TONELENGTH(3))) "+
                "               (main "+
                "                               (DEFSTATEMENTS "+
                "                                               (OSCBANKDEF(oscb1)(2))
"+
                "
        (OSCBANKELEMENT_DEF(oscb1)(0)(SINE)) "+
                "
        (OSCBANKELEMENT_DEF(oscb1)(1)(SQUARE)) "+
                "                                               (MIXERDEF(s1))) "+
                "                               (CONSTATEMENTS "+
                "
        (OSCCON(oscb1)(BANKINDEX(0))(OSCCONARG1(0.5)(BANKINDEX))(OS
CCONARG2(1.0)(BANKINDEX))) "+
                "
        (OSCCON(oscb1)(BANKINDEX(1))(OSCCONARG1(440)(BANKINDEX))(OS
CCONARG2(oscb1)(BANKINDEX(0)))) "+
                "
        (MIXSTMT(s1)(OSCBANK(oscb1)(1))))(OUTPUT(s1)) "+
                "               ) "+
                ") ";
                expected = expected.replaceAll("\\s","");
                assertPreorder("Return Value",
                                PROGRAM,
                                expected,
                          testAST);

        }

        /* **********************************
         * 9.1 Header Section
         * *********************************/

        public void testOutputEquation() {
                System.out.println("testOutputEquation():");
                AST testAST = produce("outset", "OUTPUT = \"test.wav\"");
                System.out.println(" =>"+testAST.toStringTree());
```

```
		assertPreorder(
				"parse Output Equation", //Description
				OUTPUT,
				"(OUTPUT(test.wav))", //Expected Parser Output [pre-
order traversal]
				produce("outset", "OUTPUT = \"test.wav\"")
				);
	}

	public void testHead() {
		System.out.println("testHead():");
		String test = "" +
		"start header "+
		"OUTPUT = \"firsttest.wav\" "+
		"TONELENGTH = 10 "+
		"end header ";
		String expected = ""+
		"(HEAD" +
		"(OUTPUT (firsttest.wav))" +
		"(TONELENGTH (10))" +
		")";
		expected = expected.replaceAll("\\s","");
		AST testAST = produce("head", test);
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder(
				"Head",
				HEAD,
				expected,
				testAST
				);
	}

	//lenset?
	//segset?

	/* ***********************************
	 * 9.2 Main Function
	 * ***********************************/

	public void testMainFunction() {
		System.out.println("testMainFunction():");
		String test = ""+
		"start func main \n"+
		"	start def \n"+
		"		//osc o1 = \"SINE\"; \n"+
		"		//osc o2 = \"SQUARE\"; \n"+
		"			oscbank oscb1 = 2; \n"+
		"			oscb1[0]=\"SINE\"; \n"+
		"			oscb1[1]=\"SQUARE\"; \n"+
		"			mixer s1; \n"+
		"		end def \n"+
		"\n"+
		"		start connect \n"+
		"			oscb1[0](0.5,1.0); \n"+
		"			oscb1[1](440,oscb1[0]); \n"+
		"			s1 = oscb1[1]; \n"+
		"		end connect \n"+
		"		 \n"+
		"		OUTPUT = s1; \n"+
		"end func main \n";
		String expected = ""+
		"(main "+
		"			(DEFSTATEMENTS "+
		"					(OSCBANKDEF "+
		"							(oscb1)(2)) "+
		"
	(OSCBANKELEMENT_DEF(oscb1)(0)(SINE)) "+
		"
	(OSCBANKELEMENT_DEF(oscb1)(1)(SQUARE)) "+
		"							(MIXERDEF(s1))) "+
		"			(CONSTATEMENTS "+
		"
	(OSCCON(oscb1)(BANKINDEX(0))(OSCCONARG1(0.5)(BANKINDEX))(OS
CCONARG2(1.0)(BANKINDEX))) "+
		"
	(OSCCON(oscb1)(BANKINDEX(1))(OSCCONARG1(440)(BANKINDEX))(OS
CCONARG2(oscb1)(BANKINDEX(0)))) "+
		"					(MIXSTMT(s1)(OSCBANK(oscb1)(1)))) "+
		"			(OUTPUT(s1)) "+
		") ";
		expected = expected.replaceAll("\\s","");
		AST testAST = produce("main", test);
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder(
			"Main Function",
			MAIN,
			expected,
			testAST
			);
	}
	/*
	//main?

	public void testConditionCon() {
		System.out.println("testConditionCon():");
		AST testAST = produce("ifcon",
"	if(x==1 || SEG==2 || SEG==5)"+
"	{"+
"		ob1[x](440, 0.5);\n"+
"}" +
"\n");
		System.out.println(" =>"+testAST.toStringTree());
		assertPreorder("parse ifcon rule",
				LITERAL_if,

	"(if(||(||(==(x)(1))(SEGSTMT(==)(2)))(SEGSTMT(==)(5)))(CONSTATEMENTS
(OSCBCON(ob1)(x)(440)(0.5))))",
				testAST);

//		testAST = produce("ifcon",
//"	if(x==1 || SEG==2 || SEG==5)"+
//"	{"+
//"		ob1[x](440, 0.5); //apostrophes differentiate variables"+
//"	}"+
//"	else"+
//"	{"+
//"		ob1[x](220*x, ob1[x-1]);"+
//"	}");
//		System.out.println(" =>"+testAST.toStringTree());
//		assertPreorder("parse ifcon rule",
//				OSCCON,
//				"(OSCCON(o1)(440)(e1))",
//				testAST);
//
//
//		testAST = produce("forcon",
//"for(x=1; x<3; x++) //for loop featuring .size feature (returns # of osc in"+
//"{ //oscbank"+
//"	if(x==1 || SEG==2 || SEG==5)"+
//"	{"+
//"		ob1[x](440, 0.5); //apostrophes differentiate variables"+
//"	}"+
//"	else"+
//"	{"+
//"		ob1[x](220*x, ob1[x-1]);"+
//"	}"+
//"}");
//			System.out.println(" =>"+testAST.toStringTree());
//			assertPreorder("parse forcon rule",
//					OSCCON,
//				"(OSCCON(o1)(440)(e1))",
//				testAST);
	}
	*/



}
```

AaslTreeWalkerTester.java

```
/*
 * AaslTreeWalkerTester.java
 * Albert Tsai
*/

package aasl.tests;

import java.io.StringReader;

import org.norecess.antlr.ParserTestCase;
import org.norecess.antlr.TestableParser;
import org.norecess.antlr.TreeParserTestCase;
//TreeBuilderTestCase ?
import aasl.*;
import aasl.intermediateTypes.*;
import antlr.*;
import antlr.collections.AST;


public class AaslTreeWalkTester extends TreeParserTestCase implements
AASLParserTokenTypes {
	//@Override
	protected TokenStream makeLexer(String input) {
		AASLLexer ret = new AASLLexer(new StringReader(input));
		return ret;
	}
	protected TestableParser makeParser(TokenStream lexer) {
		AASLParser parser = new AASLParser(lexer);

	parser.getASTFactory().setASTNodeClass("org.norecess.antlr.LineNumberAST")
;
```

```
		return parser;
	}
	protected TreeParser makeTreeParser() {
		return new AaslAntlrWalker();
	}

	/*
	 *
=================================================================
====
	 *						TREE WALKER TESTS
	 *
=================================================================
====
	 * http://cs.calvin.edu/curriculum/cs/382/jdfrenscompilers/parser.html
	 * more info @ http://antlr-testing.sourceforge.net/
	 *
	 * IR: Intermediate Representation
	 *
	 * assertEquals(IR Class, build(treeProduction, produce(parseProduction, input)));
   *
   * IR Class: IR class from aasl.intermediateTypes
   * build(treeProduction, AST): Given an AST, run it against the specified Tree
Production.
   * treeProduction: Tree Walker production rule to test against.
   *
   * produce(parseProduction,input): Produces an AST, using the following parameters.
   * parseProduction: Parser production rule to run against the input.
   * input: String representation of character stream to run parser against.
   *
   * Summary:
   * In short, you are seeing if the Tree Production takes the specified AST and correctly
   * outputs the IR class.
	 */
	public void testInteger() throws RecognitionException {
		//assertEquals(new AaslInt(5), build("expr", produce("expr","5")));
		AaslAntlrWalker walker = new AaslAntlrWalker();
		assertEquals(new AaslInt(5), walker.expr(produce("expr","5")));

	}
}
```

CodeGenTest1.java

```
/*
```

```
* CodeGenTest1.java
*Rob Katz
*/

package aasl.tests;

import aasl.intermediateTypes.*;
import aasl.*;

import java.util.*;

public class CodeGenTest1 {
        public static void main(String args[]){
                Vector v = new Vector();
                Vector funcv = new Vector();
                v.add("TestCycle1");
                v.add(new AaslFloat(3));
                v.add(new AaslInt(1));

                AaslOscillator2 osc1 = new AaslOscillator2("osc1", //name
                                "SAW",
        //waveform
                                new AaslFloat(2.0f),            //freq
                                new AaslFloat(0.5f),            //amp
                                0,
        //central freq
                                1
        //segments
                                );

                AaslEnvelope env1 = new AaslEnvelope("env1");
                env1.addElement(new AaslFloat(0f), new AaslFloat(0f));
                env1.addElement(new AaslFloat(.2f), new AaslFloat(1f));
                env1.addElement(new AaslFloat(1f), new AaslFloat(0f));

                AaslOscillator2 osc2 = new AaslOscillator2("osc2", //name
                                "SINE",
        //waveform
                                osc1,                                   //freq
                                env1,                                   //amp
                                440,                                    //central freq
                                1
        //segments
                                );

                AaslOscillator2 osc3 = new AaslOscillator2("osc3", //name
                                "SINE",
          //waveform
                                new AaslFloat(500f),            //freq
                                new AaslFloat(1f),                          //amp
                                500,                                                  //central freq
                                1
          //segments
                                );

                    //osc1.freq[0] = osc3;

                    AaslMixer2 mix = new AaslMixer2("testmixer1", 1);
                    mix.addElement(osc2);
                    mix.addElement(osc3);
                    AaslSegArray seg_array = new AaslSegArray(1, null);
                    seg_array.allTrue();
                    mix.setAsOutput(seg_array);

                    v.add(mix);
                    v.add(osc1);
                    v.add(env1);

                    CodeGenerator cg = new CodeGenerator(v, funcv);
          }
}
```

FuncTable.java

```
/*
* FuncTable.java
*Albert Tsai
*/

package aasl.tests;

import aasl.AASLParser;

public class FuncTable {
      static void run(String name, AASLParser p) {
            try {
            if(name.equals("funcdef"))
                  p.funcdef();
            else if(name.equals("oscdef"))
                  p.oscdef();
            else if(name.equals("defsec"))
                  p.defsec();
            else if (name.equals("consec"))
                  p.consec();
```

```
                else if (name.equals("osccon"))
                        p.osccon();
                else if (name.equals("outputeq"))
                        p.outputeq();
                else {
                        System.err.println("Attempted to execute non-
existent"
                                        + " parsing rule");
                        System.exit(1);
                }

                } catch (Exception e) {
                        System.err.println("Function Table: " + name + ": " +
e);
                }
                return;
        }

}
```

InFilter.java

```
/*
* InFilter.java
*Albert Tsai
*/

/*
 * File: InFilter.java
 * Purpose: Filter out certain files, based on a regular
 * expression and other rules. Currently, filter out:
 * Files ending in ".out"
 * Directories
 * CVN(can be excluded by above?)
 * Returns:
 * Notes:
 * The path resolutions needed to figure out if a file is a
 * directory or not is unfortunate, but necessary.
 * File(string) always figures that string is a file whose
 * root directory is ../AASL, NOT test. That's why you need
 * to use dir.
 */

package aasl.tests;

import java.io.FilenameFilter;
import java.io.File;
import java.io.IOException;
```

```
public class InFilter implements FilenameFilter {

        public boolean accept(File dir, String name) {
                //I will get nailed by Win/Unix directory issues for this
                String s = dir.toString() + "/" + name;
                File f = new File(s);

                if( name.matches(".*\056out$") || f.isDirectory() )
                        return false;
                else
                        return true;

        }

}
```

OutFilter.java

```
/*
* OutFilter.java
*Albert Tsai
*/

package aasl.tests;

import java.io.FilenameFilter;
import java.io.File;

public class OutFilter implements FilenameFilter {

        public boolean accept(File dir, String name) {
                //Match: "blahblah.out"
                if( name.matches(".*\056out$") )
                        return true;
                else
                        return false;

        }
}
```

TestList.java

```
/*
* TestList.java
```

```
*Albert Tsai
*/
package aasl.tests;

/*********************************************************
 *
 * Purpose: Given a directory containing files such as:
 *
 * Test/
 * test1
 * test1.out
 * test2
 * test2.out
 * ...
 *
 * and so on, run each file against the lexer/parser. Compare the output (a parse tree in
 * the form of a String) against the expected output, which is contained in the .out files.
 * It is assumed that the output of <name> will be compared against <name>.out
 *
 * Returns: True of all tests successful, false if otherwise
 *
 * Notes: In order to get ANTLR v2.7.x to output the parse tree you need to do a few
 * special things. See: http://www.antlr.org/article/parse.trees/index.tml
 * Specifically,
 * 1. The Parser has to extend from a different base class. This can be done by
 * using the declaration: class TinyCParser extends
Parser("antlr.debug.ParseTreeDebugParser")
 * 2. Secondly, you have to compile the grammer with the -traceParser option. You can
 * do this either in the command line, or in ANTLR Grammar Options in Eclipse, when
 * you right click on the grammar file.
 *
 *********************************************************/

import java.io.*;
import java.util.Arrays;
import antlr.ParseTree;

import antlr.CharBuffer;
import antlr.Parser;

import java.util.regex.*;

import aasl.*;

class TestList {
	File dir;					// Directory containing all the test
files
	File[] inlist, outlist;		// List of test grammars, expected parse trees.
	//CharBuffer L; //odd
	//Parser P;

	public TestList(String f) {
		// Load Test directory
		dir = new File("." + "/" + f);
		inlist = dir.listFiles(new InFilter());
		outlist = dir.listFiles(new OutFilter());
		//sorting empty directories is bad?
		Arrays.sort(inlist);
		Arrays.sort(outlist);
		//System.out.println("TestList created");
		//System.out.println(inlist.toString());
		//System.out.println(outlist.toString());
	}

	private boolean verify() {
		if(inlist.length == 0 || outlist.length == 0) {
			System.err.println("No files to look at!");
			return false;
		}

		// Make sure all test files have testfiles.out (which contains the expected
parse tree)
		if(inlist.length != outlist.length) {

			System.err.println("Can't proceed with test. There is not an equal
number of" +
			"<test> and <test>.out files");
			return false;
		}
		//Verify same names in the same order. Now can just proceed through
both lists.
		for(int i = 0; i < inlist.length; i++) {
			//getPath() worked, getName() failed.
			String e = inlist[i].getPath();
			String t = inlist[i].getPath() + ".out";
			if( t.compareTo(outlist[i].getPath()) != 0 ) {
				System.err.println("Can't proceed with test, misnamed
files.");
				System.err.println("Orphan file: " + e);
				return false;
			}
		}
		return true;
	}


	/**
	 * 1. Load Directory (done by Constructor)
	 * 2. Run Lexer/Parser on each file
	 * 3. Compare generated parse tree with expected parse tree
	 * 4. Return true if ALL are successful, false otherwise.
	 * @return
	 * @throws Exception
	 */
	public boolean execute(String rule) throws Exception {
		boolean result = true;
		if( ! verify() )
			return false;

		//Isolate the try's, maybe split into two?
		for( int i = 0; i < inlist.length; i++ ) {
			//Do I need this?
			//String fin = inlist[i].getCanonicalPath();
			String fin = inlist[i].getPath();
			String fout = fin + ".out";
			try {
				//Pass this to the lexer
				BufferedReader in = new BufferedReader(new FileReader(
fin ));
				BufferedReader out = new BufferedReader(new
FileReader( fout ));
				//BufferedWriter out2 = new BufferedWriter(new
FileWriter( "wtf" ));
				System.out.print("Looking at: " + fin + " ");
				StringBuffer sb = new StringBuffer();
				String inline;

				/*
				 * readLine
				Reads a line of text. A line is considered to be terminated
by
				any one of a line feed ('\n'), a carriage return ('\r'),
				or a carriage return followed immediately by a linefeed.
				Returns:
   A String containing the contents of the line, **not including** any line-termination
   characters, or null if the end of the stream has been reached
				 */
				//can't guarantee spaces between elements...potential
problem.
				while((inline = out.readLine()) != null ) {
					sb.append(inline + " ");
				}
				// Run Parser against each file
				// Can't really split the try's, because in is used here.
				AASLLexer lexer = new AASLLexer( in );
				AASLParser parser = new AASLParser(lexer);

				//parser.funcdef();
				//parser.oscdef();
				FuncTable.run(rule, parser);
//				ParseTree tree = parser.getParseTree();
				ParseTree tree = null;
				System.out.println("parse tree: "+tree.toStringTree());

				String output = tree.toStringTree();
				//String sb = " ( <program> ( <declaration> ( <variable> (
<type> int ) ( <declarator> i ) ; ) ) EOF )";
				output = output.trim();
				//doesn't trim tabs (actually this shouldn't matter)
				//tri//dir = new File("." + "/" + f);m only cuts the leading
and trailing spaces.
				String expected = sb.toString().trim();
				expected = expected.replace("\t", ""); //do something about
the tabs.
				expected = expected.replaceAll("  +", " ");

				System.out.println("Expected: " + expected);

				if( output.compareTo(expected) != 0 ) {
					System.out.println("FAIL: Parse trees did not
match");
					result = false;
				}
				else {
					System.out.println("SUCCESS");
				}
				in.close();
				out.close();
				//out2.close();

				//We want to differentiate between IOExceptions, and
parser exceptions
			} catch(IOException e) {
				System.err.println("Failed to read[" + fin + " | " + fout +"]: 
" + e);
			} catch(Exception e) {
				System.out.println("FAIL: Parser/Lexer bombed on file");
				System.out.println("exception: "+e);
				result = false;
				System.err.flush();
			}
		}
		return result;
	}
}
```

ParserTest.java

```
/*
* ParserTest.java
*Albert Tsai
*/

package aasl.tests;

/**
 * File: ParserTest
 * Purpose: Run JUnit 4.x tests, using TestList. TestList is a directory of
 * test files and test outputs, along it methods to run a specified lexer/parser
 * against them. If all files pass (see TestList for further description)
 * then the unit test for the directory passes. Thus these are not "unit"
 * tests, per se, but I believe that this test architecture will prove itself
 * the easiest/fastest to use.
 *
 * Anticipated Use: The directories contain "classes" of tests (although
 * it could contain an arbitrary collection of files). So one JUnit test would be on a
 * directory containing various permutations of declarations in our language, another
 * directory would contain permutations of statements, etc. With this setup, adding 
another
 * test would be trivial - you just add another test file, named N, and the result you
 * expect, N.out. You wouldn't have to add another JUnit test or any additional code.
 *
 * Notes:
 * This uses JUnit 4.x, which takes advantage of Java 5.0 language constructions
 * (specifically, the @Label stuff).
 */
```

```
//Now change to JUnit 3.8 =*(

import java.io.*;
//import junit.framework.JUnit4TestAdapter;
//import static org.junit.Assert.*;
//import org.junit.*;

import junit.framework.TestCase;

public class ParserTest extends TestCase {
	//List procedure for generating parse trees that this thing can use.
	private boolean result = false;

	/*
	@Before public void setup() {
		result = false;
	}
	*/

	//Test for one directory, whatever the hell is in it. Abstraction!
	/*
	public void testDirectory() {
		if(true)
			assertEquals(true, true);
			return;
		try {
			// Assume current directory, look for this subdirectory.
			// ** Files are run with the root directory of
			// /Analog Additive Synthesis Language !!! ** Keep this in mind.
			TestList t1 = new TestList("test");
			result = t1.execute("program");
			assertEquals(result, true);
		} catch (Exception e){
			//Catch exceptions? Would this be useful?
			//blah
		}
	}
	*/
	/*
	public void testDefinitions() {
		try {
		TestList t1 = new TestList("test/oscdef");
			result = t1.execute("oscdef");
			assertEquals(result, true);
		} catch (Exception e) {
			System.err.println("testDefinitions()" + e);
```

```
        }
}


public void testDefsec() {
        try {
        TestList t1 = new TestList("test/defsec");
                result = t1.execute("defsec");
                assertEquals(result, true);
        } catch (Exception e) {
                System.err.println("testDefsec() " + e);
        }
}*/

public void testDefsec() {
        if(true) {assertEquals(true, true); return;} //cheap way to short circuit test
        String name = "defsec";
        try {
        TestList t1 = new TestList("test/" + name);
                result = t1.execute(name);
                assertEquals(result, true);
        } catch (Exception e) {
                System.err.println("test_" + name + "(): " + e);
        }
}

public void testConsec() {
        if(true) {assertEquals(true, true); return;}
        String name = "consec";
        try {
        TestList t1 = new TestList("test/" + name);
                result = t1.execute(name);
                assertEquals(result, true);
        } catch (Exception e) {
                System.err.println("test_" + name + "(): " + e);
        }
}

public void testOsccon() {
        if(true) {assertEquals(true, true); return;} //cheap way to short circuit test
        String name = "osccon";
        try {
        TestList t1 = new TestList("test/" + name);
                result = t1.execute(name);
                assertEquals(result, true);
        } catch (Exception e) {
                System.err.println("test_" + name + "(): " + e);
            }
    }

    public void testOutputeq() {
            if(true) {assertEquals(true, true); return;} //cheap way to short circuit test
            String name = "outputeq";
            try {
            TestList t1 = new TestList("test/" + name);
                result = t1.execute(name);
                assertEquals(result, true);
            } catch (Exception e) {
                System.err.println("test_" + name + "(): " + e);
            }
    }

    public void testFuncdef() {
            if(false) {assertEquals(true, true); return;}
            String name = "funcdef";
            try {
            TestList t1 = new TestList("test/" + name);
                result = t1.execute(name);
                assertEquals(result, true);
            } catch (Exception e) {
                System.err.println("test_" + name + "(): " + e);
            }
    }

    /*
    @After public void teardown() {
            result = false;
    }
    */

}
```

Aasl Sample Programs

chords.aasl

```
/*
*     chords.aasl
*
*     contains functions that generate major chords rooted at a
frequency equal
*           to the integer argument
*/
```

```
start header
OUTPUT = "chords"
TONELENGTH = 2
end header

start func majorSine0(int x)
      start def
            int tones=3;
            oscbank ob1=tones;
            int i;
            for(i=0; i<tones; i=i+1)
                  {ob1[i]="SINE";}
      end def
      start connect
            ob1[0](x,1.0);
            ob1[1](1.260*x,1.0);
            ob1[2](1.498*x,1.0);
      end connect

      OUTPUT=ob1[0]+ob1[1]+ob1[2];
end func majorSine0

start func majorSaw0(int x)
      start def
            int tones=3;
            oscbank ob1=tones;
            int i;
            for(i=0; i<tones; i=i+1)
                  {ob1[i]="SAW";}
      end def
      start connect
            ob1[0](x,1.0);
            ob1[1](1.260*x,1.0);
            ob1[2](1.498*x,1.0);
      end connect

      OUTPUT=ob1[0]+ob1[1]+ob1[2];
end func majorSaw0



start func main
      start def
            mixer m1;
      end def

      start connect
            m1 = majorSaw0{440};
      end connect

      OUTPUT = m1;
end func main
```

envtest.aasl

```
/*
*     envtest1.aasl
*
*     Run this and view the output in an audio editor for a good view
of what
*           an envelope attached to an oscillators amplitude input will
do.
*/

start header
      OUTPUT = "envtest1"
      TONELENGTH = 3
      SEGMENTS = 2
end header

start func main
      start def
            oscbank oscb1 = 2;
            if(SEG==1){
                  oscb1[0]="SINE";
            }
            else{
                  oscb1[0]="SAW";
            }

            oscb1[1]="SQUARE";

            env e1 = { (0.0,0.3) (0.2,0.0) (0.4,1.0) (0.8,0.8)
(1.0,0.0) };

            mixer s1;
      end def

      start connect
            oscb1[0](2,1.0);
            oscb1[1](oscb1[0]@440,e1);

            s1 = oscb1[1];
      end connect

      OUTPUT = s1;
end func main
```

envtest2.aasl

```
/*
*     envtest2.aasl
*
*     fun with envelopes and integer variables
*/

start header
OUTPUT = "envtest2"
TONELENGTH = 3
SEGMENTS = 2
```

```
end header

start func main
      start def
            int x=2;
            int y=1;
            int z=x+y;
            oscbank oscb1 = 2;
            if(SEG==1){
                  oscb1[0]="SINE";
            }
            else{
                  oscb1[0]="SAW";
            }

            oscb1[1]="SQUARE";

            osc o1="REVSAW";

            env e1 = { (0.0,0.3) (0.2,0.0) (0.4,1.0) (0.8,0.8) };

            mixer s1;
      end def

      start connect
            oscb1[0](z,y);
            oscb1[1](oscb1[0]@440,e1);
            o1(440,1);

            s1 = 0.4*oscb1[1] + 0.5*o1;
      end connect

      OUTPUT = s1;
end func main
```

factortest1.aasl

```
/*
*     factortest1.aasl
*
*     View the output's spectral plot for a good representation of
oscillator
*            addition and multiplication
*/


start header
OUTPUT = "factortest1"
TONELENGTH = 3
end header

start func o3(int x)
      start def
            osc o3="SINE";
      end def
      start connect
            o3(x,1.0);
      end connect

      OUTPUT = o3;
end func o3



start func main
      start def
            int x=2000;
            int y=3*x;
            osc o1 = "SINE";
            osc o2 = "SINE";

      end def
      /*
      * ha ha comments
      */
      start connect
            o1(x,1.0);
            o2(2*x,1.0);
      end connect

      OUTPUT = 200*o1 + 50*o2 + o3{20000};
end func main
```

forlooptest1.aasl

```
/*
*     forlooptest1.aasl
*
*     the first for loop makes every oscillator in the bank a sine wave
*     the second gives them all mathematically related values for
frequency and
*           amplitude.  This is useful for building complex
waveforms(ie. a square
*           wave) out of simple components (ie. sine waves)
*/

start header
      OUTPUT = "forlooptest1"
      TONELENGTH = 3
end header

start func main
      start def
            int x;
            int y;
            int z=5;
            oscbank oscb1 = z;
            for(x=0;x<z;x=x+1){
                  oscb1[x]="SINE";
            }

            mixer s1;
      end def

      start connect
            for(x=2;y<z;x=x*2){
                  oscb1[y](440*x,2.0/x);  //diferentiates between 2
(int) and 2.0 (float)
                  s1 = s1 + oscb1[y];
                  y=y+1;
            }

      end connect

      OUTPUT = s1;
end func main
```

forlooptest2.aasl

```
/*
*     forlooptest2.aasl
*
*     combines loops with mixer assignment
*/


start header
OUTPUT = "forlooptest2"
TONELENGTH = 3
end header

start func main
      start def
            int x;
            int y;
            int z=5;
            oscbank oscb1 = z;
            for(x=0;x<z;x=x+1){
                  oscb1[x]="SINE";
            }

            mixer s1;
      end def

      start connect
            for(x=1;y<z;x=x+2){
                  oscb1[y](440*x,1.0/x);
                  s1 = s1 + oscb1[y];
                  y=y+1;
            }

      end connect

      OUTPUT = s1;
```

```
end func main
```

helloworld.aasl

```
/*
*     helloworld.aasl
*
*     Generates a simple sine wave at 440 Hz and full volume.
*/

start header
OUTPUT = "helloworld"
TONELENGTH = 1
end header

start func main
      start def
            osc o1="SINE";
      end def

      start connect
            o1(440,1.0);
      end connect

      OUTPUT = o1;
end func main
```

helloworld2.aasl

```
/*
*     helloworld2.aasl
*
*     Generates a simple sine wave at 440 Hz and full volume using an
oscillator
*           from an oscillator bank.
*/


start header
OUTPUT = "helloworld"
TONELENGTH = 3
end header

start func main
      start def
            oscbank ob1=1;
            ob1[0]="SINE";
      end def

      start connect
            ob1[0](440,1);
      end connect

      OUTPUT = ob1[0];
```

```
end func main
```

osc2osc.aasl

```
/*
*     osc2osc.aasl
*
*     Example of using oscillators to control other oscillators.  [0]
is the
*           audible oscillator. [1] modifies its frequency. [2] in turn
modifies
*           the frequency of [1].
*/

start header
OUTPUT = "osc2osc"
TONELENGTH = 12
end header

start func main
      start def
            oscbank oscb1 = 3;
            oscb1[0]="SQUARE";
            oscb1[1]="SINE";
            oscb1[2]="SAW";

            env e1 = { (0.0,0.3) (0.2,0.0) (0.4,1.0) (0.8,0.8)
(1.0,0.0) };

            mixer s1;
      end def

      start connect
            oscb1[0](oscb1[1]@1000,e1);
            oscb1[1](oscb1[2]@3,1.0);
            oscb1[2](0.4,1.0);

            s1 = oscb1[0];
      end connect

      OUTPUT = s1;
end func main
```

osc2osc2.aasl

```
/*
*     osc2osc2.aasl
*
*     Example of using oscillators to control other oscillators.  [0]
is the
*           audible oscillator. [1] modifies its frequency. [2] in turn
modifies
*           the frequency of [1].
*     Note also that not assigning an oscillator in the def section 
will set it
*           automatically to "SINE".
*/

start header
OUTPUT = "osc2osc2"
TONELENGTH = 12
end header

start func main
      start def
            oscbank oscb1 = 3;
            int x;
            oscb1[1]="SINE";
            oscb1[2]="SQUARE";

            mixer s1;
      end def

      start connect
            oscb1[0](oscb1[1]@1000,1.0);
            oscb1[1](oscb1[2]@3,1.0);
            oscb1[2](0.7,1.0);

            s1 = oscb1[0];
      end connect

      OUTPUT = s1;
end func main
```

segtest1.aasl

```
/*
*     segtest1.aasl
*
*     Simple program involving oscillator banks and segments.  [0] is 
an
*           oscillator that controls [1]'s amplitude.  Run this and 
look at the
*           output in an audio editor for a clear representation of 
amplitude
*           control and segments.
*/



start header
OUTPUT = "segtest1"
TONELENGTH = 3
SEGMENTS = 2
end header

start func main
      start def
```

```
            oscbank oscb1 = 2;
            if(SEG==0){
                  oscb1[0]="SINE";
            }
            else{
                  oscb1[0]="SAW";
            }

            oscb1[1]="SQUARE";

            mixer s1;
      end def

      start connect
            oscb1[0](2,1.0);
            oscb1[1](oscb1[0]@440,oscb1[0]);

            s1 = oscb1[1];
      end connect

      OUTPUT = s1;
end func main
```

segtest2.aasl

```
/*
*     segtest2.aasl
*
*     More fun with segments.
*/

start header
OUTPUT = "segtest2"
TONELENGTH = 3
SEGMENTS = 4
end header

start func main
      start def
            oscbank o1 = 1;
            int x=234;
            if(SEG==1 || SEG==2){
                  o1[0]="SAW";
            }
            else{
                  o1[0]="SINE";
            }

      end def
      /*
      * ha ha comments
      */
      start connect
      if(SEG==1 || SEG==3 && x==233 || SEG==3){
            o1[0](x,1.0);
```

```
		}
	if(SEG==0 || SEG==2){
		o1[0](400,0.5);
	}
	/*if(SEG==3){
		o1[0](2500,1.0);
	}*/
	end connect

	OUTPUT = o1[0];
end func main
```

segtest3.aasl

```
/*
*	segtest3.aasl
*
*	Even more fun with oscillators.
*/

start header
OUTPUT = "segtest1"
TONELENGTH = 3
SEGMENTS = 2
end header

start func main
	start def
		//osc o1 = "SINE";
		//osc o2 = "SQUARE";
		oscbank oscb1 = 2;
		if(SEG==0){
			oscb1[0]="SINE";
		}
		else{
			oscb1[0]="SAW";
		}

		oscb1[1]="SQUARE";

		mixer s1;
	end def

	start connect
		oscb1[0](2,1.0);
		oscb1[1](oscb1[0]@440,oscb1[0]);

		s1 = oscb1[1];
	end connect

	OUTPUT = s1;
end func main
```

userDefFunction1.aasl

```
/*
*     userDefFunction.aasl
*
*     This features a user defined function that takes an integer and
creates
*           two oscillators who's frequency values depend on the
argument.  The
*           function is used like a single oscillator in assignment of
mixer s1.
*/

start header
      OUTPUT = "userDefFunction1"
      TONELENGTH = 3
      SEGMENTS = 2
end header


//function definition...takes one integer argument
start func udefFunc(int x)
      start def
            osc oUdef1="SINE";
            osc oUdef2="SINE";
            mixer s1;
      end def

      start connect
            oUdef1(x, 0.75);
            oUdef2(0.75*x, 0.75);

            s1 = oUdef1 + oUdef2;
      end connect

      OUTPUT = s1;
end func udefFunc


start func main
      start def
            int x=2;
            int y=1;
            int z=x+y;
            oscbank oscb1 = 2;
            if(SEG==1){
                  oscb1[0]="SINE";
            }
            else{
                  oscb1[0]="SAW";
            }
            oscb1[1]="SQUARE";

            osc o1="REVSAW";
            mixer s1;
      end def
      start connect
            oscb1[0](z,y);
            oscb1[1](oscb1[0]@440,1.0);
            o1(440,1);

            s1 = udefFunc{1000} + 2*o1 + 4*oscb1[1];
      end connect

      OUTPUT = s1;
end func main


/*


*/
```

userDefFunction2.aasl

```
start header
      OUTPUT = "userDefFunction2"
      TONELENGTH = 3
      SEGMENTS = 2
end header


//function definition...takes one integer argument
start func udefFunc(int x)
      start def
            osc oUdef1="SAW";
            mixer s1;
      end def

      start connect
            oUdef1(0.6*x, 0.75);

            s1 = oUdef1;
      end connect

      OUTPUT = s1;
end func udefFunc

start func udefFunc(float x)
      start def
            osc oUdef1="SINE";
            mixer s1;
      end def

      start connect
            oUdef1(x, 0.75);

            s1 = oUdef1;
      end connect
      OUTPUT = s1;
end func udefFunc


start func main
      start def
            int x=2;
            int y=1;
            int z=x+y;
            oscbank oscb1 = 2;
            if(SEG==1){
                  oscb1[0]="SINE";
            }
            else{
                  oscb1[0]="SAW";
            }

            oscb1[1]="SQUARE";

            osc o1="REVSAW";

            env e1 = { (0.0,0.3) (0.2,0.0) (0.4,1.0) (0.8,0.8)
(1.0,0.0) };

            mixer s1;
      end def

      start connect
            oscb1[0](z,y);
            oscb1[1](oscb1[0]@440,e1);
            o1(440,1);

            s1 =udefFunc{500.} + udefFunc{500};
      end connect

      OUTPUT = s1;
end func main


/*


*/
```

Zelda.aasl

```
start header
OUTPUT = "zelda"
TONELENGTH = 17
SEGMENTS = 64
end header

start func main
      start def
```

```
            //central octave

            float a0=440; float as0=1.059*a0; float b0=1.122*a0; float
c0=1.189*a0;
            float cs0=1.260*a0; float d0=1.335*a0; float ds0=1.414*a0;
            float e0=1.498*a0; float f0=1.587*a0; float fs0=1.682*a0;
            float g0=1.782*a0; float gs0=1.888*a0;

            //one up
            float a1=2*a0; float as1=2*as0; float b1=2*b0; float
c1=2*c0;
            float cs1=2*cs0; float d1=2*d0; float ds1=2*ds0; float
e1=2*e0;
            float f1=2*f0; float fs1=2*fs0; float g1=2*g0; float
gs1=2*gs0;
            //two up
            float a2=2*a1; float as2=2*as1; float b2=2*b1; float
c2=2*c1;
            float cs2=2*cs1; float d2=2*d1; float ds2=2*ds1; float
e2=2*e1;
            float f2=2*f1; float fs2=2*fs1; float g2=2*g1; float
gs2=2*gs1;
            //three up
            float a3=2*a2; float as3=2*as2; float b3=2*b2; float
c3=2*c2;
            float cs3=2*cs2; float d3=2*d2; float ds3=2*ds2; float
e3=2*e2;
            float f3=2*f2; float fs3=2*fs2; float g3=2*g2; float
gs3=2*gs2;
            //four up
            float a4=2*a3; float as4=2*as3; float b4=2*b3; float
c4=2*c3;
            float cs4=2*cs3; float d4=2*d3; float ds4=2*ds3; float
e4=2*e3;
            float f4=2*f3; float fs4=2*fs3; float g4=2*g3; float
gs4=2*gs3;
            //...and one down
            float am1=220; float am1s=1.059*am1; float bm1=1.122*am1;
            float cm1=1.189*am1; float csm1=1.260*am1; float
dm1=1.335*am1;
            float dsm1=1.414*am1; float em1=1.498*am1; float
fm1=1.587*am1;
            float fsm1=1.682*am1; float gm1=1.782*am1; float
gsm1=1.888*am1;
            //...and two down
            float am2=110; float asm2=1.059*am2; float bm2=1.122*am2;
            float cm2=1.189*am2; float csm2=1.260*am2; float
dm2=1.335*am2;
            float dsm2=1.414*am2; float em2=1.498*am2; float
fm2=1.587*am2;
            float fsm2=1.682*am2; float gm2=1.782*am2; float
gsm2=1.888*am2;

            int x=32;
            int i;

            oscbank oscb1 = 2;
            oscb1[0]="SINE";
            oscb1[1]="SAW";

            oscbank harm= 2;
            harm[0]="SQUARE";
            harm[1]="SQUARE";

            env env1 = { (0.0,0.0) (0.5,0.0) (0.5,1.0) };

            mixer s1;
      end def

      start connect
      oscb1[1](440,1);
      for(i=0; i<2; i=i+1){
            if( SEG==1+(x*i) || SEG==5+(x*i) || SEG==25+(x*i))
                  {oscb1[0](g0,1.0);}
            if(SEG==29+(x*i))
                  {oscb1[0](a1,1.0);}
            if(SEG==7+(x*i) || SEG==30+(x*i))
                  {oscb1[0](b1,1.0);}
            if(SEG==2+(x*i) || SEG==9+(x*i) || SEG==13+(x*i) ||
SEG==17+(x*i) ||
                   SEG==21+(x*i) || SEG==26+(x*i))
                  {oscb1[0](c1,1.0);}
            if(false)
                  {oscb1[0](cs1,1.0);}
            if(SEG==14+(x*i) || SEG==22+(x*i) || SEG==28+(x*i) ||
SEG==31+(x*i))
                  {oscb1[0](d1,1.0);}
            if(SEG==4+(x*i))
                  {oscb1[0](ds1,1.0);}
            if(SEG==3+(x*i) || SEG==10+(x*i) || SEG==15+(x*i) ||
SEG==24+(x*i) ||
                         SEG==27+(x*i))
                  {oscb1[0](e1,1.0);}
            if(SEG==18+(x*i) || SEG==23+(x*i))
                  {oscb1[0](f1,1.0);}
            if(false)
                  {oscb1[0](fs1,1.0);}
            if(SEG==0+(x*i) || SEG==12+(x*i))
                  {oscb1[0](g1,1.0);}
            if(SEG==20+(x*i))
                  {oscb1[0](gs1,1.0);}
            if(SEG==8+(x*i) || SEG==11+(x*i) || SEG==16+(x*i) ||
SEG==19+(x*i))
                  {oscb1[0](a2,1.0);}
            if(false)
                  {oscb1[0](as2,1.0);}
            if(SEG==6+(x*i))
                  {oscb1[0](b2,1.0);}
      }

            for(i=0; i<2; i=i+1){
            if(SEG==2+(x*i) || SEG==6+(x*i) || SEG==10+(x*i) ||
SEG==26+(x*i))
                  {oscb1[1](gm2,1.0);}
            if(SEG==30+(x*i))
                  {oscb1[1](am1,1.0);}
            if(SEG==4+(x*i) || SEG==28+(x*i) || SEG==31+(x*i))
                  {oscb1[1](bm1,1.0);}
            if(SEG==0+(x*i) || SEG==4+(x*i) || SEG==8+(x*i) ||
SEG==12+(x*i) ||
                  SEG==13+(x*i) || SEG==18+(x*i) || SEG==22+(x*i))
                  {oscb1[1](cm1,1.0);}
            if(SEG==14+(x*i))
                  {oscb1[1](dm1,1.0);}
            if(SEG==15+(x*i) || SEG==24+(x*i))
                  {oscb1[1](em1,1.0);}
            if(SEG==16+(x*i) || SEG==20+(x*i))
                  {oscb1[1](fm1,1.0);}
            if(SEG==1+(x*i) || SEG==3+(x*i) || SEG==5+(x*i) ||
SEG==7+(x*i) ||
                  SEG==9+(x*i) || SEG==11+(x*i) || SEG==27+(x*i))
                  {oscb1[1](gm1,1.0);}
            if(SEG==17+(x*i) || SEG==19+(x*i) || SEG==21+(x*i) ||
SEG==23+(x*i) ||
                  SEG==25+(x*i))
                  {oscb1[1](c0,1.0);}
            }

            for(i=0; i<2; i=i+1){
                  if(SEG==0+(x*i) || SEG==1+(x*i) || SEG==2+(x*i) ||
SEG==3+(x*i))
                        {harm[0](g2,env1);
                         harm[1](e2,env1);}
                  if(SEG==4+(x*i) || SEG==5+(x*i) || SEG==6+(x*i) ||
SEG==7+(x*i))
                        {harm[0](ds2,env1);
                         harm[1](b2,env1);}
                  if(SEG==8+(x*i) || SEG==9+(x*i) || SEG==10+(x*i) ||
SEG==11+(x*i))
                        {harm[0](b3,env1);
                         harm[1](ds2,env1);}
                  if(SEG==12+(x*i) || SEG==13+(x*i) || SEG==14+(x*i) ||
SEG==15+(x*i))
                        {harm[0](a3,env1);
                         harm[1](e2,env1);}
                  if(SEG==16+(x*i) || SEG==17+(x*i) || SEG==18+(x*i) ||
SEG==19+(x*i))
                        {harm[0](g2,env1);
                         harm[1](e2,env1);}
                  if(SEG==20+(x*i) || SEG==21+(x*i) || SEG==22+(x*i) ||
SEG==23+(x*i))
                        {harm[0](a2,env1);
                         harm[1](f2,env1);}
                  if(SEG==24+(x*i) || SEG==25+(x*i))
                        {harm[0](d3,env1);
                         harm[1](f2,env1);}
                  if(SEG==26+(x*i) || SEG==27+(x*i))
                        {harm[0](c3,env1);
                         harm[1](f2,env1);}
                  if(SEG==28+(x*i) || SEG==29+(x*i) || SEG==30+(x*i) ||
SEG==31+(x*i))
```

```
                        {harm[0](g2,env1);
                         harm[1](e2,env1);}
            }

            s1 = oscb1[0] + 0.2*oscb1[1] + 0.1*harm[0] + 0.1*harm[1];
      end connect

      OUTPUT = s1;
end func main
```