CMajor

A Music Production Language

Andrew O'Reilly	Stephanie Huang	Jonathan Sun	Laura Tang
ajo2119	syh2115	jys2124	lt2510

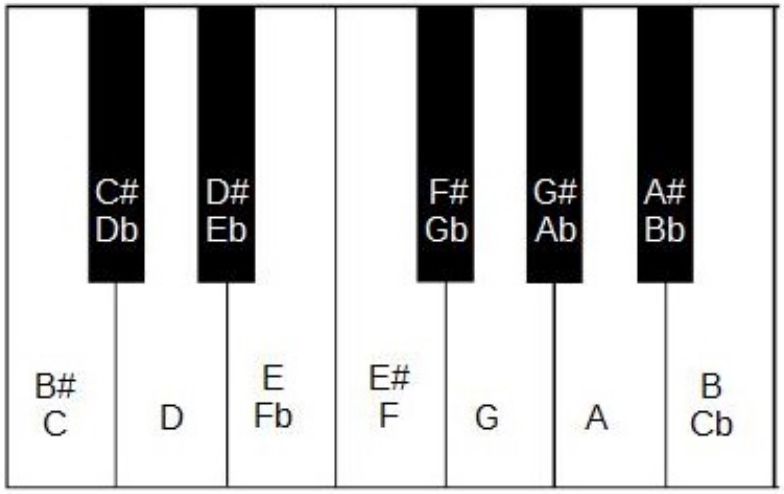
Contents

1 Introduction

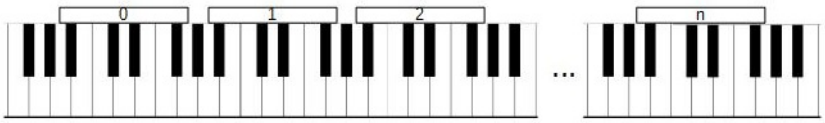
CMajor is a procedural, imperative language used to create musical compositions for playback on a MIDI device. It provides a set of types for abstracting time and frequency components of Western music, as well as a notation for referencing frequencies and pitches commonly employed. CMajor outputs the results of a composition to CSV bytecode, later to be interpreted by a Java program and rendered via MIDI playback. In addition to data types which correspond to the duration and pitch aspects of sound, it provides structured types which allow a programmer composer to organize pitches into sequences and to layer them into chords and phrases, giving them control over harmonic properties of musical composition as well as its melodic ones. Users of the language may also take advantage of familiar programming constructs such as loops and conditional statements, allowing them to easily repeat precomposed phrases, reuse previously composed structures, and conditionally alter the course of a composition based on number of repetitions or whatever conditions they choose to supply. CMajor possesses a C-style syntax, consisting of lists of expressions separated by semicolons, each of which return types that can be operated upon according to the rules of the language. Programmers may additionally write their own functions to modify pitches or return composed elements.

1.1 Background

Perhaps one of the most fascinating aspects of music is that its generation entails complex mathematical calculations, and that these calculations can be made by a performer and perceived by listeners regardless of their mathematical abilities. Further, the twelve tone western system further abstracts these calculations by classifying different frequencies as named "pitches" and uses a system of fractions to describe durations. In this system, frequencies, here called pitches, are given a letter and optionally a modifier to raise their frequency. Each key on a piano is made to strike and therefore vibrate, a different string, each tuned to one of these frequencies. The keys of a piano therefore provide a useful diagram for the arrangement of named pitches according to their frequency:



The frequencies named above increase from left to right. The pitch that corresponds to the key that would appear immediately to the right of the "B / Cb" key would be named "B# / C" along with the one that appears furthest to the left in the above diagram. This not-pictured pitch has a frequency equal to twice that of the pitch to the far left; the one after it has a frequency equal to twice that of the the one corresponding to the next pictured key (C# / Db), and so on. The perceived difference between any two pitches is called an interval, and in the case where the perceived difference is between a pitch and its corresponding one of doubled frequency, the interval is called an octave. Pitches an octave apart share a certain auditory quality and are easily identified, and so the pattern of keys on a piano repeats itself:



In CMajor, pitches are identified according to the naming scheme employed in the first picture (a pitch with two names may be referred to by either one) and by their octave number, with higher numbers referring to octaves with higher frequencies. CMajor further takes inspiration from Western music in its classification of pitch duration as a basis for rhythm. All durations are stored as a pair of integers, which represent the numerator and denominator of a rational number. The actual duration in milliseconds depends upon the number of beats per minute, and further upon the definition of a beat Western music tends to be flexible in this regard, but here a beat is defined as 1/4, or a quarter note, and the beats per minute is set to 120:

Numerator	Denominator	Length (ms)
1	1	2000
1	2	1000
1	4	500
1	8	250
3	8	750
1	16	125
3	16	375
1	12	166.6666666667

Above are a few durations that may be constructed, although any pair of integers may be used to do so. By using a combination of a pitch and a duration, a note may be constructed, and a sequence of notes may be used to create a song. Notes may also be played simultaneously to create harmonies and interlocking rhythms that add complexity to a composition. Two other important aspects of Western musical composition, timbre and volume, were not considered for this project and are left to a future implementation.

2 Langauge Tutorial

2.1 Installation & Compilation

To install CMajor, run make from the root directory after extracting the tar file. This will build and compile all the components needed for the CMajor compiler. Next, write your CMajor composition in a file ending in .cmaj. Example CMajor programs can be found in the tests/directory, as well as the demo/directory.

Compile your CMajor program by running: ./cmajor scale.cmaj

This will generate two output files: an out.csv file and a play.out file. out.csv is an intermediate "bytecode" file, comparable to .class files generated by the Java compiler. play.out is the executable file (a generated shell script), which can be executed with the following command in order to play the music composed:

```
./play.out
```

Alternatively, out.csv can be manually played by executing the following command using the included CSVPlayer:

```
java CSVPlayer out.csv
```

2.2 Compose with CMajor

Every CMajor file (.cmaj) is a self contained piece of music that contains the functions, sequences of statements, and control structures necessary to describe and play that piece of music. All musicians write music by composing. They listen to music by playing. This gives us our two most important functions in CMajor: compose() and play().

Every CMajor file must contain a compose() function, and if the piece is to be played, must call the built-in play() function. A simple CMajor program to play the single note middle C is shown below:

```
/*
 * play middle C
 */
int compose() {
  // call play on a note literal
  play(($C, (1,4));
}
```

After compiling, we will get an out.csv and play.out file. These files are only generated when play() is called within the compose() function.

CMajor uses C-like syntax with function return types in the function declarations, explicit typing, brackets enclosing blocks of code, and semicolon line endings.

In the example above, we use the two methods of commenting. Inline/single line comments using // and multiline comments enclosed by /* */. We also create a note literal, which is represented by a tuple of pitch and duration. Pitch literals are simple note letters preceded by $. Octave and sharps and flats can also be utilized (ex. $C#4 is a C# in octave 4, or the note a half step above middle C).

CMajor supports common control structures such as ifelse statements and for loops. You can also write your own functions. Example syntax is shown below:

```
// for loops
int i; // note that i is initialized outside the loop
for (i = 0; i < 4; i = i + 1) {
```

```
  // code here...
}
// ifelse statements
if (i == 0) {
  // code here...
}
else {
  // code here...
}
// custom functions have the following syntax
returntype function_name(paramtype parameter1, paramtype parameter2, ...) {
  // code here...
  return foo;
}
```

CMajor has some special types that make music writing easier. These include pitch and duration types, as well as structural types like notes, chords, phrases, and scores.

One key feature of CMajor is the special music related operators that we have. In particular, we feature the layer $\hat{()}$ operator, which allows for creation of notes, chords, phrases, and scores from layers of types such as pitches and durations (to form a phrase), or multiple phrases to form a score. We also have an array concatenation operator (++) and repeater operator for replication of structural types such as notes or chords (**).

The Language Reference Manual is a self-contained document in the following pages.

C-Major Language Reference Manual

1. Expressions

An expression is a series of tokens that return a value. They consist of one or more literals and zero or more operators. Expressions are grouped according to their operators (if present) and evaluated according to operator precedence. One or more expressions may be combined at terminated with a semicolon (;) to form an expression statement, or separated by commas (,) to form a list for use in function calls. A list of expressions of variable size make up the body of blocks, which are delimited by braces ({ }). An array of expressions separated by the comma (,) character may be used to populate an array.

stmt_list → stmt_list stmt | ε

stmt → expr;

actuals_list → expr | actuals_list, expr

Basic expressions consist of one or more identifiers (see Lexical Conventions) and zero or more operators. An identifier may be a literal or a variable.

expr → expr op expr

Assignment expressions assign the value returned by an expression to an identifier The type of value returned by the expression must match the type of the variable represented by the identifier.

expr → id = expr

Function calls consist of an identifier followed by an open parenthesis, followed by an expression array. The return value of the expression is the return value of the function.

expr → id(expr_array)

2. Data Types

2.1 Primitive Types

There are two primitive types in C-Major, int and pitch, upon which all other types in the language are built.

2.1.1 Int

Represents a whole number.

2.1.2 Pitch

Pitch represents a musical pitch, typically an integer that maps to an index on the piano keys (0-88). It is stored internally as an integer. The default pitch is 40 (C4).

2.2 Non-Primitive/Structural Types

2.2.1 Array

An array type has the format t[] where t is a type that specifies the type of all elements of the array. Thus, all elements of an array of type t[] must themselves have type t. Note that t itself may be an array type.

Arrays *can* be initialized as an array *literal* of type *literals*:

```
int[] array = [1,2,3,4,5];
```

2.2.2 Tuple

A tuple is a pair of elements within parenthesis separated by a comma. Each element can be a different type.

2.2.3 Duration

A duration is tuple of integers. The ratio of the first element to the second element represents the fraction of a whole note the associated pitch will play.

2.2.4 Note

A note is a tuple consisting of a pitch and a duration. The pitch must be in the left element.

(pitch, duration)

2.2.5 Chord

A chord is a tuple wherein the left element is an array of pitches, and the right element is a duration type element. All pitches in the array will be played for the duration specified by the second element.

(pitch[], duration)

2.2.6 Phrase

A phrase is an array of chords. This would represent a single line or voice of music in a piece. Every note will start and end individually; there are no overlaps. A second voice should be designated with a separate phrase. A variable of type phrase may be initialized to or otherwise assigned the value of an expression whose type is a chord array.

chord[]

2.2.7 Score

A Score is an array of phrases. Each element points to a single phrase which would represent the multiple voices of a single piece. A variable of type score may be initialized to or assigned the value of an expression whose type is an array of phrases.

phrase[]

3. Operators

3.1 Assignment Operator =

As previously stated, the assignment operator is denoted by the equals sign =.

3.2 Comparison Operators

Comparison operators are used to test for equality or inequality between identifiers or literals. A expression consisting of a comparison operator and two other expressions return an integer type whose value is 1 where the assertion is true and 0 where it is false. All comparison operators test the value of their identifiers. The return type of each expression being operated on by comparison operators must be the same.~~,~~ The greater-than, greater-than-or-equal-to, less-than, or less-than-or-equal-to operators (>, >=, <, and <=, respectively) may be used with the following types:

```
int
pitch
duration
```

The equality and inequality operators (== and !=, respectively) may additionally be used with the note type.

Production rule	Description
expr → expr == expr	Evaluates to 1 if the return values of the expressions in the production body are equivalent, and 0 otherwise.
expr → expr != expr	Evaluates to 1 if the return values of the expressions in the production body are not equivalent, and 0 otherwise.
expr → expr > expr	Evaluates to 1 if the expression on the left is greater in return value than the return value of expression on the right, and 0 otherwise.
expr → expr < expr	Evaluate to 1 if the expression on the right is greater in return value than the return value of expression on the right, and 0 otherwise.
expr → expr >= expr	Evaluates to 1 if the expression on the left is greater in return value than the expression on the right, or if the return values of the expressions are equal, and 0 otherwise.

expr → expr <= expr	Evaluates to 1 if the expression on the right is greater in return value than the expression on the left, or if the return values of the expressions are equal, and 0 otherwise.

The inequality of integers is evaluated according to the standard ordering of integers from negative infinity to infinity. In evaluations of pitch types, their inequality is evaluated according to their frequency or the position of their corresponding keys on a piano-- pitches that correspond to keys towards the right end of the piano are greater than pitches that correspond to keys on the left. The inequality of durations is evaluated according to a standard ordering of rational numbers from 0 to infinity.

3.3 Arithmetic Operators

Arithmetic operators are binary operators and consist of addition (+), subtraction (-), multiplication (*), and division (/). The return type of expressions involving arithmetic operators depends upon the return type of the expressions in the operation. Addition and subtraction are commutative.

Operator	Symbol	Left expression type	Right expression type	Return value
Addition	+	int	int	The sum of the two integers.
		pitch	int	A pitch raised the number of half steps indicated by the integer.
		dur	int	A duration. The integer is converted to a fractionally equivalent duration. The durations are then added according to fractional arithmetic. (1,2) + 1 = (3,2)
		dur	dur	The sum of the two durations according to fractional arithmetic,

				reduced to its least possible denominator.
Multiplication	*	int	int	The product of the two integers.
		dur	int	The product of the fractional value of the duration and the integer, reduced to the least possible denominator. (1,4) * 2 yields (1,2).
		dur	dur	The fractional product of the two durations. (1,4) * (1,2) yields (1,8).
Subtraction	-	int	int	The difference between the left integer and the right integer.
		pitch	int	A pitch lowered by the number of half steps specified by the integer expression.
		dur	int	A duration whose length is the the result of the fractional subtraction of right integer converted to a fraction from the fractional value of the left duration expression. If the result is negative, the absolute value is returned. (5,4) - 1 = (1,4)
		pitch	pitch	An integer representing the difference between the two pitches, in scale positions.
		chord	pitch	A chord with the right-expression pitch removed, if it was present.
		dur	dur	A duration whose length is equal to the fractional subtraction of the right duration from the left. (1,2) - (1,4) = (1,4)
		note	dur	A note whose duration is equal to the subtraction of the right duration from the duration of the left note expression.

		chord	dur	A chord whose duration is equal to the subtraction of the right duration from the duration of the left note expression.
Division	/	int	int	A duration whose numerator is equal to the left integer and whose denominator is equal to the right.
		dur	int	A duration whose fraction is equal to the fractional division of the fractional component of the left expression by the integer value of the right expression. (1,2) / 2 = (1,4)
		note	int	A note whose duration is equal to the division of the duration of the note in the left expression divided by the integer value of the right expression, as described above.
		chord	int	A chord whose duration is equal to the division of the duration of the chord in the left expression divided by the integer value of the right expression, as described above.
		int	dur	A duration whose fractional component is equal to the fractional division of the integer by the the fractional value of the duration. 1 / (1,2) = (2,1)
		dur	dur	Fractional division of durations. (1,2) / (1,4) = (2,1)

	note	dur	A note whose duration is equal to the fractional division of the left expression's duration component by the right expression's duration.
	chord	dur	A chord whose duration is equal to the fractional division of the left expression's duration component by the right expression's duration.
	dur	note	A note whose duration is equal to the fractional division of the left duration by the duration component of the note in the right expression.
	dur	chord	A chord whose duration is equal to the fractional division of the left duration by the duration component of the note in the right expression.
	dur	chord	A chord whose duration is equal to the fractional division of the left duration by the duration component of the note in the right expression.

3.4 Repeater Operator - **

Supplying an expression or any type followed by the repeater operator (**) and a subsequent integer yields an array of size equal to the given integer with each element containing the return value of the expression:

expr → expr ** int

3.5 Concatenation Operators (+, ++)

When used exclusively with notes, chords, and phrases, the + symbol is used as a concatenation operator. The use of the + operator with any combination of notes, chords, and phrases returns a phrase type.

expr → expr + expr

The left expression is appended to the beginning of the right within the resulting phrase. All notes and chords are then intended to be read and/or played from left to right.

The ++ concatenation operator is used for array concatenation and always returns an array of the base type of its operands. One or both operands may be an array whose base type matches the base type of the other. The result is an array wherein the right expression is appended to the end of the left.

3.6 Layer Operator (^)

The layer operator is used to create musical structures wherein pitches are played simultaneously. It is a binary operator and its behavior is only defined for the pitch, note, chord, phrase, and score types.

expr → expr ^ expr

A pitch may be layered with a duration to form a note. An array of n pitches may be layered with an array of n durations to return an array of n notes, wherein the i^{th} note of the resulting array consists of the pitch at index i in the pitch array and the duration at index i in the duration array. Pitches may also be layered with chords, and in this instance a chord is returned with the pitch added. In all other cases a score is returned. When rendered, the arguments are synchronized by their beginning; if one argument has a longer total duration than the other, it continues playing after the shorter argument has completed. The layer operator is commutative.

3.7 Operator Associativity and Precedence

The layer operator is applied first, followed by the arithmetic operators -in the standard order of *, /, -, +. Boolean operators are applied next, followed by the repeater operator, the array concatenation operator, and finally the assignment operator.

4. Lexical Conventions

4.1 Comments

Comment syntax is similar to Java. Single line comments are preceded by //. Multiline comments are enclosed with /* and */. For example:

```
// Single line comment

/*
* Multiline
* comment
* here
*/
```

4.2 Identifiers

An identifier names functions and variables and consists of a sequence of alphanumeric characters and underscores (_) in the set `[ ‘a’-‘z’ ‘A’-’Z’ ‘_’ ‘0’-’9’ ]`. Identifiers are case-sensitive and must begin within a character within the set `[ ‘_’ ‘a’-’z’ ‘A’-’Z’ ]`.

4.3 Keywords

The following keywords are reserved:

chord	dur	else
false	for	if
int	note	null
phrase	pitch	play
print	return	score
true	void	

4.4 Constants/Literals

Integer literals

Integer literals are of type int and are of the form [‘0’-’9’]

Pitch Literals

Pitch literals are of type pitch and are of the form ‘$’ [‘A’-’G’] [‘#’ ‘b’]? [‘0’-’9’]?
The capital letter corresponds to the note name, ‘#’ and ‘b’ denote sharp or flat, and the integer denotes which octave the note is in. If ‘#’ or ‘b’ is omitted, a natural pitch is assumed. If an octave integer is omitted, octave 4 is assumed, or the octave of the set key (see more on setting keys later on). For example, $C4 denotes C in octave 4, or middle C.

A rest literal is a specific pitch literal that represents a rest. (No pitch.) It is represented as $R

Duration Literals

A duration literal is of type dur and is a 2-tuple of integers that correspond to note durations used in music. It is of the form ‘(‘ [‘1’-’9’], [‘1’-’9’]+ ’)’.
For example, a quarter note can be represented as the duration literal (1,4).

Note Literals

A note literal is of type note and is a 2-tuple of pitch and duration of the form ‘(‘ (‘$’ [‘A’-’G’] [‘#’ ‘b’]? [‘0’-’9’]? | “$R”) ‘,’ ‘(‘ [‘1’-’9’], [‘1’-’9’]+ ’)’ ‘)’

Chord Literals

A chord literal is of type chord and is a 2-tuple of an array of pitches and duration. It is of the form ‘(‘ ‘[‘ (‘$’ [‘A’-’G’] [‘#’ ‘b’]? [‘0’-’9’]?)* | “$R” ‘]’ ‘,’ ‘(‘ [‘1’-’9’], [‘1’-’9’]+ ’)’ ‘)’

4.5 Separators

Separators separate tokens and expressions. White space is a separator. Other separators are tokens themselves:

```
( ) { } [ ] ; , . < >
```

4.6 White Space

White space consists of the space character, tab character, and newline character. White space is used to separate tokens and is ignored other than when used to separate tokens. White

space is not required between operators and operands or other separators. Any amount of white space can be used where one space is required.

5. Statements

5.1 Expression Statements

Any expression can become a statement by terminating it with a semicolon.

5.2 Declaration and Initialization Statements

Giving a type name keyword followed by an identifier terminated with a semicolon yields a statement that allocates memory for a variable of the given type. Optionally, the assignment operator may be supplied followed by an expression prior to the semicolon in order to initialize the variable to a value. The value to which the variable is initialized is the return value of the expression to the right of the assignment operator. As with the assignment expression, the type of the variable and the type of the value to which it is initialized must match.

5.3 if/else

An if / else statement has the following structure:

```
if (expr) {
   stmt_list
}
else if (expr) {
   stmt_list
}
else {
   stmt_list
}
```

The expression in parentheses must evaluate to true or false. If true, then the if block is executed. Otherwise, the statement is tested. The else block is executed when no conditional expression evaluates to true.

5.4 for

A for statement (for loop) has the following structure:

```
for (asn; expr1; expr2) {
  stmt_list
}
```

First, *asn* is evaluated. *asn* is traditionally an assignment expression. Next, *stmt_list* is evaluated if *expr1* evaluates to true. *expr2* is executed after *stmt_list*, and the condition in *expr1* is checked again. This repeats until *expr1* evaluates to false and the for statement is exited.

5.5 return expr;

The return statement evaluates *expr* and returns program control to the function that called it, and returns the evaluated value of *expr* into the higher level function. The type of *expr* must be the same as declared in the function definition.

6. Functions

6.1 Defining Functions

Function definitions have the form:

type declarator compound-statement

The *type* specifies the return type. A function can return any type. The declarator in a function declaration must specify explicitly that the declared identifier has a function type; that is, it must be of the form

direct-declarator (*expr_array*)

The form and its parameters, together with their types, are declared in its parameter type list; the declaration-list following the function's declarator must be absent. Each declarator in the parameter type list must contain an identifier.

A *parameter-type-list* is a list of expressions separated by commas. The parameters are understood to be declared just after beginning of the compound statement constituting the function's body, and thus the same identifiers must not be redeclared there (although they may, like other identifiers, be redeclared in inner blocks). An example:

```
int max(int a, int b) {
        if (a > b) return a;
        else return b;
}
```

Here int is the declaration specifier; max(int a, int b) is the function's declarator, and { … } is the block giving the code for the function.

6.2 Calling Functions

A function call is an identifier followed by parentheses containing a possibly empty, comma-separated list of assignment expressions which constitute the arguments to the function, or an expression array. The term *argument* is used for an expression passed by a function call; the term *parameter* is used for an input object (or its identifier) received by a function definition, or described in a function declaration.

In preparing for the call to a function, a copy is made of each argument; all argument-passing is strictly by value. A function may change the values of its parameter objects, which are copies of the argument expressions, but these changes cannot affect the values of the arguments. The types of parameters are explicit and are part of the type of the function - this is the function prototype. The arguments are converted, as if by assignment, to the types of the corresponding parameters of the function's prototype. The number of arguments must be the same as the number explicitly described parameters. Recursive calls to any function are permitted.

6.3 The *play* Function

The identifier *play* is reserved to let the compiler make MIDI calls in Java. *Play* takes either a *score* type expression or *phrase* type expression. It returns an integer: 0 on success, 1 for failure.

6.4 The *compose* Function

Every *C-Major* program must define the reserved identifier *compose*. The expression bound to *compose* is evaluated and its value is the value of the *C-Major* program itself. That is, when a *C-Major* program is compiled and run, the expression bound to *compose* is evaluated and the result is converted to a value of type score or int. If a definition for *compose* is not included, or the expression bound to it does not evaluated to *score*, a compile-time error will occur.

4 Project Plan

4.1 Project Process

4.1.1 Planning

We had a one hour meeting every Monday with all members in attendance. These meetings were led by our manager Andrew, and we discussed project milestones including what we would accomplish within the following week and made updates to the CMajor language design. Further into the project, we averaged two to three meetings per week where we would discussed implementation that overlapped between members and any debugging that needed to be done.

4.1.2 Specification

For the Proposal and LRM, we outlined what was to be included during our weekly meetings, and assigned sections to different members. Led by Jonathan, who composed a larger chunk of the reports and made the final transposition into LaTeX, each member wrote their own sections and proofread the documents individually, making edits as needed.

4.1.3 Development

While developing our language, Andrew and Stephanie were the primary authors of the scanner, parser, and analyzer/code generator code. Usually coding was done individually, and then reviewed by peers after submitting pull requests to a master branch of our project on a GitHub repository. Language features such as arrays, control structures, operators, and types were divided up and implemented independently of each other, so that we had a working compiler early on and simply expanded it out by adding features. This made testing much easier as well.

4.1.4 Testing

Testing was accomplished using the test suite written by Laura (described later on in detail in this report). We made sure to test during the development process, especially when implementing new features. Before and after merging every pull request that implemented a new feature, we would run the test suite to make sure any conflicts or bugs were resolved. We wrote sample test cases specific to the features being implemented while implementing those features.

4.2 Style Guide

All code was implemented using Unix line endings and spaces for indentation. Function bodies and other nested blocks of code were indented with two spaces. Lines were broken and indented two spaces when lines were longer than 84 characters. Match statements were similarly implemented, with the -> operator on the same line as the match case and subsequent lengthy code on following lines.

4.3 Project Timeline

Date	Milestone
Sep. 30, 2015	Proposal Due
Oct. 23, 2015	LRM Outlined
Oct. 25, 2015	LRM Drafted
Oct. 26, 2015	LRM Proofread
Oct. 26, 2015	LRM Due
Oct. 31, 2015	Scanner and Parser
Nov. 12, 2015	Semantic Analyzer
Nov. 16, 2015	Hello World Due
Dec. 16, 2015	Scanner and Parser Completed
Dec. 17, 2015	Semantic Analyzer Completed
Dec. 18, 2015	Testing Completed
Dec. 18, 2015	Presentation Due
Dec. 21, 2015	Code Cleanup
Dec. 22, 2015	Final Report Due

Commit Graph:

4.4 Roles and Responsibilities

Andrew O'Reilly	Manager
Stephanie Huang	System Architect
Jonathan Sun	Language Guru
Laura Tang	Testing Suite

4.5 Development Environment

CMajor has been tested and built in both OS and Windows 8 (running cygwin and an Ubuntu virtual machine) environments. Git was used for version control. Text editors used include vim, Sublime Text, and Notepad++. Most of the CMajor language (scanner, parser, compiler) was written in OCaml, utilizing features such as ocamllex. In order to generate sound written by

CMajor programs, we also used Java 7 with the javax.sound.midi library. Testing suites, Makefiles, and final output files utilize bash/shell scripting.

4.6 Project Log

(See following pages)

Hash	Author	Date	Comment
4b4447c	Andrew	Tue Dec 22	Merge pull request #60 from phanieste/compiler
fb8d3f8	Stephanie	Tue Dec 22	writes executable play.out file to play out.csv
19a331d	Stephanie	Tue Dec 22	add authors to source code files
3812bda	Jonathan	Tue Dec 22	Compile Java with make. Move java files to root
3d16371	Andrew	Tue Dec 22	Convert tabs to spaces
17e8d93	Andrew	Tue Dec 22	Clean up comments
71921c1	Jonathan	Sat Dec 19	gitignore *.toc
b31d5f0	Jonathan	Sat Dec 19	Additional .gitignores
eba4e78	Jonathan	Sat Dec 19	Initial final report latex template.
0da6088	Andrew	Fri Dec 18	Lengthen shepard
81bd6fc	Stephanie	Fri Dec 18	implement layer for score with phrase
7711edf	Andrew	Fri Dec 18	Add demos
592c2f2	Stephanie	Fri Dec 18	play works with single note
f2b3513	Andrew	Fri Dec 18	Move up precedence of repeater operator
7c8df08	Stephanie	Fri Dec 18	fix compile warnings with incomplete match case
ff588ae	Stephanie	Fri Dec 18	fixes csv output and pitch ordering
623fbe4	Stephanie	Fri Dec 18	add note+note=phrase implementation
d2c5e80	Laura	Fri Dec 18	fix failure report test.sh
84ee8cb	Laura	Fri Dec 18	update failure report
bcdca10	Laura	Fri Dec 18	tests folder renaming
6bc98d3	Stephanie	Fri Dec 18	fix runtime errors in play
f509204	Laura	Fri Dec 18	renamed tests
df8fb9f	Stephanie	Fri Dec 18	implements multi-note play
36a0671	Laura	Fri Dec 18	test.sh updated with failure reports
3c34776	Laura	Fri Dec 18	test.sh updated to compare to .outs
b34507c	Laura	Fri Dec 18	added .outs
8f2be0d	Andrew	Thu Dec 17	Add tests
2c4fde4	Andrew	Thu Dec 17	Implement if/else
17fadce	Andrew	Thu Dec 17	Add tests
15988fa	Andrew	Thu Dec 17	Implement for loops
cf66fb8	Andrew	Thu Dec 17	Add tests
79d88dc	Andrew	Thu Dec 17	Fix block statement processing
3906eec	Stephanie	Thu Dec 17	fixes issue #42
b1c1512	Andrew	Thu Dec 17	Implement changes in execute.ml
9256b4d	Andrew	Thu Dec 17	Fix field names in compile.ml
ba4c86b	Andrew	Thu Dec 17	Fix typos in function definitions
837c8b1	Andrew	Thu Dec 17	Add missing arguments
ac49964	Andrew	Thu Dec 17	Implement find_var for environments and symtabs
9180c7d	Andrew	Thu Dec 17	Implement update_var function

```
71da4c2     Laura       Thu Dec 17 :modified test.sh testing script
f473fbd     Andrew      Thu Dec 17 :More descriptive exception message
a319b9f     Andrew      Thu Dec 17 :Compiles without warnings
7efabf0     Andrew      Thu Dec 17 :Resolve last compile.ml warning
ff99449     Andrew      Thu Dec 17 :Resolve most match/unused case warnings in comp
aee1d0f     Andrew      Thu Dec 17 :Clear several match warnings
d99f5c8     Andrew      Thu Dec 17 (Merge pull request #43 from goodtimefamilyband/
7b2cda4     Andrew      Thu Dec 17 (Add subtraction test
d62bbf0     Andrew      Thu Dec 17 (Change subtraction ops to return correct types
a18fd56     Andrew      Thu Dec 17 (Suppress unmatched case warnings
a3aca81     Andrew      Thu Dec 17 (Implement -
5bd4571     Andrew      Wed Dec 16 :Add tests
00fce05     Andrew      Wed Dec 16 :Implement * (multiply)
e0510a7     Andrew      Wed Dec 16 :Implement **
17d5053     Andrew      Wed Dec 16 :Merge branch 'phanieste-compiler-pitchfix' into
75d1b64     Andrew      Wed Dec 16 :Additional fixes
8d3186b     Stephanie   Wed Dec 16 :remove sign from pitch literal
5db8a33     Stephanie   Wed Dec 16 :implement + operator for notes
b90e2b7     Stephanie   Tue Dec 15 :implement + operator for dur + int
2dfcfe1     Stephanie   Tue Dec 15 :implement + operator for pitch
43a1d88     Stephanie   Tue Dec 15 :Fix merge conflicts between compiler and compil
85d2377     Jonathan    Tue Dec 15 :Cleanup & whitespace issues.
6502b4d     Andrew      Tue Dec 15 :Add tests for >=
ced3054     Andrew      Tue Dec 15 :Implement <=
71eeca8     Andrew      Tue Dec 15 :Implement <
5a45cf0     Andrew      Tue Dec 15 :Implement >=
111b024     Andrew      Tue Dec 15 :Add tests for >
f90761b     Andrew      Tue Dec 15 :Update gcd function to handle 0
   8.49E+34 Andrew      Tue Dec 15 :Fix typo
d0d27df     Andrew      Tue Dec 15 :Modify gcd function to handle negative numbers
db0ae91     Andrew      Tue Dec 15 :Implement > operator in syntactically correct f
46ddd69     Andrew      Tue Dec 15 :Add dur_sub function for comparators
e3357f2     Andrew      Tue Dec 15 (Add tests for != operator
cd3953a     Andrew      Tue Dec 15 (Implement != operator
13ca5ad     Andrew      Tue Dec 15 (Add test for ==
62f90dd     Andrew      Tue Dec 15 (Implement == operator
fa4dff6     Andrew      Tue Dec 15 (Fix missing Vdecl case for arrays
c355d51     Andrew      Tue Dec 15 (Update tests
0dba605     Andrew      Tue Dec 15 (Fix compile errors
bb366ea     Andrew      Mon Dec 14 :Fix line endings
```

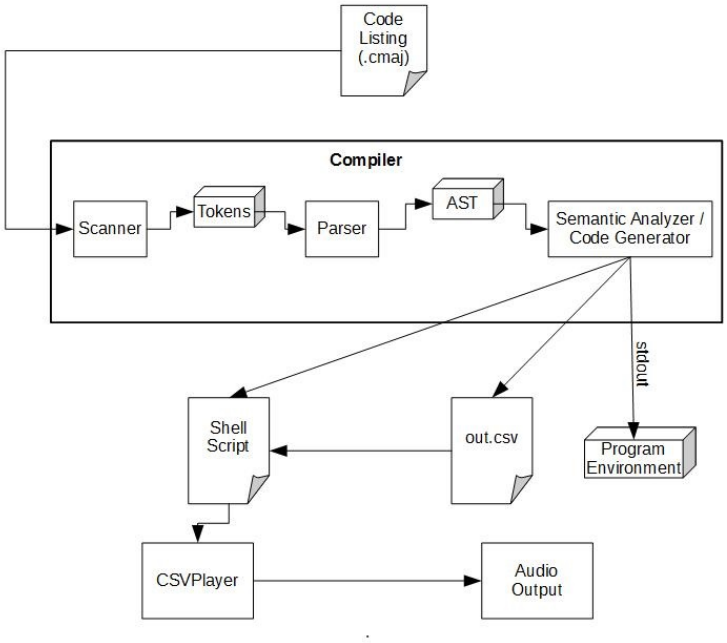
```
a30746e     Andrew      Mon Dec 14 2Fix line endings
d08c422     Laura       Mon Dec 14 1JONATHAN: Implement rests (silent notes). Remov
7e8ee7f     Jonathan    Sat Dec 12 (Clean newlines
6a49a48     Jonathan    Sat Dec 12 (CSVPlayer can play multiple lines independently
03aab2e     Andrew      Fri Dec 11 1Change line endings to Unix format
d9c8058     Andrew      Fri Dec 11 1Convert line endings to Unix
4d06091     Andrew      Fri Dec 11 1Add test and expected output for issue #32
be601d2     Andrew      Fri Dec 4 1Fix #32 parsing issues with pitch literals
dae7eb8     Andrew      Thu Dec 3 2Try using separate parser rule to fix pitch iss
b8b3942     Stephanie   Tue Dec 8 2implement ++ for concatenating two single eleme
02a77de     Stephanie   Tue Dec 8 2implement layer operator with pitch[] ^ dur[]
f4091af     Stephanie   Mon Dec 7 1implement array concatenation operator (++)
bb438be     Stephanie   Sat Dec 5 1fix bug with assigning score and phrase types
b1391bd     Stephanie   Sat Dec 5 1implement layer operator
f23005c     Stephanie   Fri Dec 4 1implement array set operation
c1a49ad     Andrew      Fri Dec 4 1Fix #32 parsing issues with pitch literals
aac4850     Stephanie   Fri Dec 4 0implement array get operation
44f6f87     Andrew      Thu Dec 3 2Try using separate parser rule to fix pitch iss
0c3fe22     Stephanie   Wed Dec 2 2implement phrases and scores
3e0e6c7     Stephanie   Wed Dec 2 2implement chords
072ac0e     Stephanie   Wed Dec 2 2implement array type checking
98b61c2     Stephanie   Wed Dec 2 2declare arrays using typename[] syntax
f33b78c     Stephanie   Wed Dec 2 1basic array literal creation and array type
6b57379     Stephanie   Mon Nov 30 Fix merge conflicts merging compiler into compi
61f3545     Andrew      Sun Nov 29 Implement Call expression
a8bdbc6     Andrew      Sun Nov 29 Add Missing_function exception
d2d2091     Stephanie   Sun Nov 29 add arrays to parser and ast
5ad5706     Stephanie   Sun Nov 29 manually merge and fix compilation errors in pa
b95a157     Andrew      Sun Nov 29 Update exec_fun to return environment
cd460df     Andrew      Sun Nov 29 Reorganize code
65b3803     Andrew      Sun Nov 29 Update toplevel test to test return
e388f18     Andrew      Sun Nov 29 Implement return statement
5e92323     Andrew      Sun Nov 29 Implement function to get string from s_type
359086b     Andrew      Sun Nov 29 Update tests to use new syntax
b86de3c     Andrew      Sun Nov 29 Add script to convert existing tests to new syn
9b16265     Andrew      Sun Nov 29 Make translate recursive to expose exec_fun fun
b8bf861     Andrew      Sun Nov 29 Update global environment data type
bb0700d     Andrew      Sun Nov 29 Remove option from globals type
8e4e288     Andrew      Fri Nov 27 Change global environment var map to allow empt
a24d2bb     Andrew      Fri Nov 27 Add global_environment type
```

```
19bc53d     Andrew      Fri Nov 27 :Fix compile syntax errors
50fe379     Andrew      Fri Nov 27 :Implement fxn call
a628d17     Andrew      Fri Nov 27 :Implement code to create global environment
    5434298 Andrew      Fri Nov 27 :Update AST program type to be two lists
24d7a14     Andrew      Fri Nov 27 (Update parser/compiler to accept top level synt
a66e496     Andrew      Sun Nov 29 :Add division test
1c48398     Andrew      Sun Nov 29 :Simplify dur fractions
d18a5cd     Andrew      Thu Nov 26 :Fix basic test syntax
cae256d     Andrew      Thu Nov 26 :Fix missed variable rename
12c0c73     Andrew      Thu Nov 26 :Update execute module to handle new types
6d6d9d1     Andrew      Thu Nov 26 :Update main executable to handle new types
f27ac50     Andrew      Thu Nov 26 :Add List.rev to block
f86b44d     Andrew      Thu Nov 26 :Update compiler to use returned values instead
e06ab57     Andrew      Thu Nov 26 (Fix syntax errors in test.cmaj
b1ea069     Andrew      Tue Nov 24 :More spacing/readability/code style
ae69b29     Andrew      Tue Nov 24 :More spacing/readability/code style
7e41e20     Andrew      Tue Nov 24 :Spacing/readability/code style
9a27498     Andrew      Tue Nov 24 :Remove excess parentheses
15b0315     Andrew      Tue Nov 24 :Implement divide operator for chords
72baa53     Andrew      Tue Nov 24 :Implement divide Binop
58838af     Andrew      Tue Nov 24 :Remove excess parens from dur_divide fxn
ab7c25f     Andrew      Tue Nov 24 :Add dur_divide function
4ed9b7c     Andrew      Tue Nov 24 :Change Pitch to 3 ints instead of 3 Ints
30f165a     Andrew      Tue Nov 24 :Change Dur to be two ints instead of 2 Ints
0789bfa     Andrew      Tue Nov 24 :Fix syntax errors
423b8cd     Andrew      Tue Nov 24 :Add Binop to expr function
3d19c29     Laura       Thu Nov 19 :made test folder and tests
e7008e0     Andrew      Wed Nov 18 :Finish play function
41c80dc     Stephanie   Wed Nov 18 :start writing play() function
f657f4b     Andrew      Wed Nov 18 :Add missing line to compile
b71d23a     Andrew      Tue Nov 17 :Add new stmt type
     368599 Stephanie   Mon Nov 16 :Fix some variable declaration errors.  * Use Li
1ee5b63     Stephanie   Mon Nov 16 :switch env to hashtable
ba0a887     Stephanie   Mon Nov 16 :attempt to fix variable declaration struggles
9e25f2a     Jonathan    Mon Nov 16 :Fix ocamldep issues.
e224c69     Stephanie   Mon Nov 16 :add execute.ml
ac7f876     Jonathan    Mon Nov 16 :Generate additional Makefile lines with ocamlde
f46cdd7     Stephanie   Mon Nov 16 :Merge pull request #8 from phanieste/compiler
18fdd1a     Stephanie   Mon Nov 16 :Fixed syntactical and logical compilation error
d8afb17     Andrew      Mon Nov 16 :Fix scanner ID/Typename conflict
```

```
94766b7     Stephanie   Mon Nov 16 :Fix compilation errors in sast with c_type to s
20a1d24     Stephanie   Mon Nov 16 (Start building out working hello world version.
b53047f     Jonathan    Sun Nov 15 :Ignore java *.class files.
d25b059     Jonathan    Sun Nov 15 :Rename 'readCSV' to 'play.' Add print statement
40d008b     Jonathan    Sun Nov 15 :Remove duplicate file.
db58345     Jonathan    Sun Nov 15 :Twinkle CSV example.
df124bf     Jonathan    Sun Nov 15 :CSVPlayer complete.
13c52f0     Stephanie   Fri Nov 13 :Fix merge conflicts in semantics.ml
a00045f     Stephanie   Fri Nov 13 :start compiler
9309fdd     Andrew      Fri Nov 13 :Resolve merge conflict in sast
70006d3     Andrew      Fri Nov 13 :Add Makefile
9cb2e60     Andrew      Fri Nov 13 :Update PITCH_SIGN token in scanner to mitigate
c5bec06     Andrew      Fri Nov 13 :Add code to read cmajor code from stdin
21f101d     Andrew      Fri Nov 13 :Fix typo in scanner
eecfe5b     Andrew      Fri Nov 13 :Fix parser compile errors
0b24a9e     Andrew      Fri Nov 13 (Remove mergetool garbage
993ca5d     Andrew      Fri Nov 13 (Fix syntax errors
a607e9f     Andrew      Fri Nov 13 (Initial implementations of exec
4f6adbf     Andrew      Fri Nov 13 (Fixing AST for temporarily simplified program s
b21223e     Andrew      Thu Nov 12 :Removing mergetool garbage
bfeccf9     Andrew      Thu Nov 12 :Fixing conflicts with origin branch
6d0e4e8     Andrew      Thu Nov 12 :Fix syntax errors in compiler
9953a3a     Andrew      Thu Nov 12 :Fix additional syntax errors in SAST
240d856     Andrew      Thu Nov 12 :Fix syntax errors
559d148     Andrew      Thu Nov 12 :Fix undefined constructor in AST
703240a     Andrew      Thu Nov 12 :Implement variable lookup and update routines
9f59f3e     Andrew      Thu Nov 12 :Add semantic types for checking
44c0b72     Andrew      Mon Nov 9 1Add main compiler file
bd55d06     Andrew      Mon Nov 9 1Change duration literal to tuple
f96c439     Andrew      Mon Nov 9 1Simplifying test prog further
76a6cff     Andrew      Mon Nov 9 1Adding test program
045f1f2     Andrew      Mon Nov 9 1Basic test grammar - no functions
db42241     Andrew      Thu Nov 12 :Adding Steph's initial SAST
c739bd5     Andrew      Mon Nov 9 1Add main compiler file
3abc520     Andrew      Mon Nov 9 1Change duration literal to tuple
874db3a     Andrew      Mon Nov 9 1Simplifying test prog further
f3d9184     Andrew      Mon Nov 9 1Adding test program
    6271431 Andrew      Mon Nov 9 1Basic test grammar - no functions
57b3aa4     Stephanie   Wed Nov 11 :began sast and semantic analysis (pitch validat
ba22901     Stephanie   Mon Nov 9 1:modify pitch literals in scanner and parser to
```

```
e4884c2     Stephanie   Sun Nov 8 1:  add precedence/associativity to fix shift/reduc
940ec73     Stephanie   Sun Nov 8 1(  fix typos in scanner and parser
f4c7181     Andrew      Tue Nov 3 1:  Add PITCH_LIT ($) token
e0307d7     Andrew      Tue Nov 3 1:  Add production rules
2ebbe18     Andrew      Tue Nov 3 1:  Add regexes for if
    1222568 Andrew      Tue Nov 3 1:  Add tokens for if
957e0b7     Andrew      Tue Nov 3 1:  Add grouping operators described in previous cc
8ae08e5     Andrew      Tue Nov 3 1:  Add return
c0eedb6     Andrew      Tue Nov 3 1:  Fix comments in parser
0822f4a     Andrew      Mon Nov 2 2:  Add program start symbol
b58bc12     Andrew      Tue Nov 3 0(  Fix syntax errors
bb9732e     Andrew      Mon Nov 2 2:  Add program start symbol
15c222b     Andrew      Sat Oct 31 :  Add/fix comments
c651053     Andrew      Sat Oct 31 :  Add some statement types to AST
66977cb     Andrew      Sat Oct 31 :  Add vdecl type for variable declarations
c700771     Andrew      Sat Oct 31 :  Fix regex quoting
ebe5c74     Andrew      Sat Oct 31 :  Add operators to AST
b035e52     Andrew      Sat Oct 31 :  Very unfinished AST
82fec91     Andrew      Sat Oct 31 (  Initial commit
```

5 Architectural Design



5.1 Components

5.1.1 Scanner

The scanner is implemented in scanner.mll and identifies language tokens using regular expressions.

5.1.2 Parser

Implemented in parser.mly. Creates the abstract syntax tree and passes it off as such to the Semantic Analyzer / Code Generator.

5.1.3 Compiler & Analyzer

Performs a dual task of semantic analysis and the storage of environment information in memory.

5.2 Interfaces

Within the compiler, all information is stored in OCaml data types and records, including a linkedlist symbol table which stores the contents of variables, a global environment type that stores global variables and a mapping of function definitions to names, and an environment type that stores a symbol table, a global environment, a return type for the current function, and its return value (if set by a return statement). Once the program has been processed, if the program calls the play() function, a CSV is output containing an intermediate form to be read by the CSVPlayer.

Andrew and Stephanie were primarily involved in the implementation of compiler components. Jonathan devised many of the language features and details. He created the CSVPlayer in java and detailed the format of the CSV files output by the compiler and read by the CSVPlayer.

6 Test Plan

6.1 Testing Phases

6.1.1 Unit Testing

Unit testing was done as language features were being completed during the coding phase of the project. Whenever a feature was added, multiple tests were run to ensure these basic blocks were parsed correctly.

6.1.2 Integration Testing

Once the unit testing was finished, the integration testing confirmed the correctness of semantic analysis and code generation.

6.1.3 System Testing

The entire endtoend testing of the language framework is the final testing phase. The CMajor compiler takes in an input program written in the language and produces an output file of the environment which is compared against the expected output of listed pitches and durations. An optional bytecode file and accompanying executable play.out file (if the play() function is present in the input program) is also generated and tested against the reference bytecode file. Finally, to test the bytecode, the play executable can be run. This utilizes the Java MIDI Player programs to produce the correct sounds, which we can listen to to ensure correct output. An output log is generated to list all the output of the test suites that are run, and a failure log is generated to list the error messages thrown for failed tests.

6.2 Examples

See following pages

rowyourboat.cmaj:

```
int compose() {
  pitch[] pitches = $C ** 3
    ++ $D ++ $E
    ++ $E ++ $D ++ $E ++ $F ++ $G
    ++ $C ** 3 ++ $G ** 3
    ++ $E ** 3
    ++ $C ** 3
    ++ $G ++ $F ++ $E ++ $D ++ $C;

  dur dot8 = (3,16);
  dur trip8 = (1,4) / 3;

  dur[] durations = (1,4) ** 2
    ++ dot8 ++ (1,16) ++ (1,4)
    ++ dot8 ++ (1,16) ++ dot8 ++ (1,16)
    ++ (1,2)
    ++ trip8 ** 12
    ++ dot8 ++ (1,16) ++ dot8 ++ (1,16)
    ++ (1,2);

  phrase mainphrase = pitches ^ durations;

  note rest = ($R,(1,1));

  int i;
  score song = newscore();

  for(i = 0; i < 4; i = i + 1) {
    int j;
    phrase round = mainphrase;
    for(j = 0; j < i; j = j + 1) {
      round = rest + round;
    }

    song = song ^ round;

  }

  play(song);

}

chord newchord(dur d) {
  note n1 = ($R,d);
  return n1 ^ $R;
}

phrase newphrase() {
  chord c1 = newchord((0,1));
  chord c2 = c1;
  return c1 + c2;
}

score newscore() {
  phrase p1 = newphrase();
  return p1 ^ p1;
}
```

-x-

.out of rowyourboat.cmaj:

```
dot8 = Dur(3,16)
durations =
Array(Dur(1,4),Dur(1,4),Dur(3,16),Dur(1,16),Dur(1,4),Dur(3,16),Dur(1,16),Dur(3,16),Dur(1,16),Dur(
1,2),Dur(1,12),Dur(1,12),Dur(1,12),Dur(1,12),Dur(1,12),Dur(1,12),Dur(1,12),Dur(1,12),Dur(1,12),Du
r(1,12),Dur(1,12),Dur(1,12),Dur(3,16),Dur(1,16),Dur(3,16),Dur(1,16),Dur(1,2))
i = Int(4)
mainphrase =
Phrase(Array(Chord(Array(Pitch(3,4)),Dur(1,4)),Chord(Array(Pitch(3,4)),Dur(1,4)),Chord(Array(Pitc
h(3,4)),Dur(3,16)),Chord(Array(Pitch(5,4)),Dur(1,16)),Chord(Array(Pitch(7,4)),Dur(1,4)),Chord(Arr
```

```
ay(Pitch(7,4)),Dur(3,16)),Chord(Array(Pitch(5,4)),Dur(1,16)),Chord(Array(Pitch(7,4)),Dur(3,16)),C
hord(Array(Pitch(8,4)),Dur(1,16)),Chord(Array(Pitch(10,4)),Dur(1,2)),Chord(Array(Pitch(3,4)),Dur(
1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(10
,
4)),Dur(1,12)),Chord(Array(Pitch(10,4)),Dur(1,12)),Chord(Array(Pitch(10,4)),Dur(1,12)),Chord(Arra
y(Pitch(7,4)),Dur(1,12)),Chord(Array(Pitch(7,4)),Dur(1,12)),Chord(Array(Pitch(7,4)),Dur(1,12)),Ch
ord(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1
,
12)),Chord(Array(Pitch(10,4)),Dur(3,16)),Chord(Array(Pitch(8,4)),Dur(1,16)),Chord(Array(Pitch(7,4
)),Dur(3,16)),Chord(Array(Pitch(5,4)),Dur(1,16)),Chord(Array(Pitch(3,4)),Dur(1,2))))
pitches =
Array(Pitch(3,4),Pitch(3,4),Pitch(3,4),Pitch(5,4),Pitch(7,4),Pitch(7,4),Pitch(5,4),Pitch(7,4),Pit
ch(8,4),Pitch(10,4),Pitch(3,4),Pitch(3,4),Pitch(3,4),Pitch(10,4),Pitch(10,4),Pitch(10,4),Pitch(7,
4),Pitch(7,4),Pitch(7,4),Pitch(3,4),Pitch(3,4),Pitch(3,4),Pitch(10,4),Pitch(8,4),Pitch(7,4),Pitch
(5,4),Pitch(3,4))
rest = Note(Pitch(-1,4),Dur(1,1))
song =
Score(Array(Phrase(Array(Chord(Array(Pitch(-1,4),Pitch(-1,4)),Dur(0,1)),Chord(Array(Pitch(-1,4),P
itch(-1,4)),Dur(0,1)))),Phrase(Array(Chord(Array(Pitch(-1,4),Pitch(-1,4)),Dur(0,1)),Chord(Array(P
itch(-1,4),Pitch(-1,4)),Dur(0,1)))),Phrase(Array(Chord(Array(Pitch(3,4)),Dur(1,4)),Chord(Array(Pi
tch(3,4)),Dur(1,4)),Chord(Array(Pitch(3,4)),Dur(3,16)),Chord(Array(Pitch(5,4)),Dur(1,16)),Chord(A
rray(Pitch(7,4)),Dur(1,4)),Chord(Array(Pitch(7,4)),Dur(3,16)),Chord(Array(Pitch(5,4)),Dur(1,16)),
Chord(Array(Pitch(7,4)),Dur(3,16)),Chord(Array(Pitch(8,4)),Dur(1,16)),Chord(Array(Pitch(10,4)),Du
r(1,2)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(3
,
4)),Dur(1,12)),Chord(Array(Pitch(10,4)),Dur(1,12)),Chord(Array(Pitch(10,4)),Dur(1,12)),Chord(Arra
y(Pitch(10,4)),Dur(1,12)),Chord(Array(Pitch(7,4)),Dur(1,12)),Chord(Array(Pitch(7,4)),Dur(1,12)),C
hord(Array(Pitch(7,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(
1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(10,4)),Dur(3,16)),Chord(Array(Pitch(8
,
4)),Dur(1,16)),Chord(Array(Pitch(7,4)),Dur(3,16)),Chord(Array(Pitch(5,4)),Dur(1,16)),Chord(Array(
Pitch(3,4)),Dur(1,2)))),Phrase(Array(Chord(Array(Pitch(-1,4)),Dur(1,1)),Chord(Array(Pitch(3,4)),D
ur(1,4)),Chord(Array(Pitch(3,4)),Dur(1,4)),Chord(Array(Pitch(3,4)),Dur(3,16)),Chord(Array(Pitch(5
,
4)),Dur(1,16)),Chord(Array(Pitch(7,4)),Dur(1,4)),Chord(Array(Pitch(7,4)),Dur(3,16)),Chord(Array(P
itch(5,4)),Dur(1,16)),Chord(Array(Pitch(7,4)),Dur(3,16)),Chord(Array(Pitch(8,4)),Dur(1,16)),Chord
(Array(Pitch(10,4)),Dur(1,2)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1,12
)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(10,4)),Dur(1,12)),Chord(Array(Pitch(10,4)
),Dur(1,12)),Chord(Array(Pitch(10,4)),Dur(1,12)),Chord(Array(Pitch(7,4)),Dur(1,12)),Chord(Array(P
itch(7,4)),Dur(1,12)),Chord(Array(Pitch(7,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord
(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(10,4)),Dur(3,1
6)),Chord(Array(Pitch(8,4)),Dur(1,16)),Chord(Array(Pitch(7,4)),Dur(3,16)),Chord(Array(Pitch(5,4))
,Dur(1,16)),Chord(Array(Pitch(3,4)),Dur(1,2)))),Phrase(Array(Chord(Array(Pitch(-1,4)),Dur(1,1)),C
hord(Array(Pitch(-1,4)),Dur(1,1)),Chord(Array(Pitch(3,4)),Dur(1,4)),Chord(Array(Pitch(3,4)),Dur(1
,
4)),Chord(Array(Pitch(3,4)),Dur(3,16)),Chord(Array(Pitch(5,4)),Dur(1,16)),Chord(Array(Pitch(7,4))
,Dur(1,4)),Chord(Array(Pitch(7,4)),Dur(3,16)),Chord(Array(Pitch(5,4)),Dur(1,16)),Chord(Array(Pitc
h(7,4)),Dur(3,16)),Chord(Array(Pitch(8,4)),Dur(1,16)),Chord(Array(Pitch(10,4)),Dur(1,2)),Chord(Ar
ray(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),
Chord(Array(Pitch(10,4)),Dur(1,12)),Chord(Array(Pitch(10,4)),Dur(1,12)),Chord(Array(Pitch(10,4)),
Dur(1,12)),Chord(Array(Pitch(7,4)),Dur(1,12)),Chord(Array(Pitch(7,4)),Dur(1,12)),Chord(Array(Pitc
h(7,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Ar
ray(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(10,4)),Dur(3,16)),Chord(Array(Pitch(8,4)),Dur(1,16))
,Chord(Array(Pitch(7,4)),Dur(3,16)),Chord(Array(Pitch(5,4)),Dur(1,16)),Chord(Array(Pitch(3,4)),Du
r(1,2)))),Phrase(Array(Chord(Array(Pitch(-1,4)),Dur(1,1)),Chord(Array(Pitch(-1,4)),Dur(1,1)),Chor
d(Array(Pitch(-1,4)),Dur(1,1)),Chord(Array(Pitch(3,4)),Dur(1,4)),Chord(Array(Pitch(3,4)),Dur(1,4)
),Chord(Array(Pitch(3,4)),Dur(3,16)),Chord(Array(Pitch(5,4)),Dur(1,16)),Chord(Array(Pitch(7,4)),D
ur(1,4)),Chord(Array(Pitch(7,4)),Dur(3,16)),Chord(Array(Pitch(5,4)),Dur(1,16)),Chord(Array(Pitch(
7,4)),Dur(3,16)),Chord(Array(Pitch(8,4)),Dur(1,16)),Chord(Array(Pitch(10,4)),Dur(1,2)),Chord(Arra
y(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),Ch
ord(Array(Pitch(10,4)),Dur(1,12)),Chord(Array(Pitch(10,4)),Dur(1,12)),Chord(Array(Pitch(10,4)),Du
r(1,12)),Chord(Array(Pitch(7,4)),Dur(1,12)),Chord(Array(Pitch(7,4)),Dur(1,12)),Chord(Array(Pitch(
7,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(3,4)),Dur(1,12)),Chord(Arra
y(Pitch(3,4)),Dur(1,12)),Chord(Array(Pitch(10,4)),Dur(3,16)),Chord(Array(Pitch(8,4)),Dur(1,16)),C
hord(Array(Pitch(7,4)),Dur(3,16)),Chord(Array(Pitch(5,4)),Dur(1,16)),Chord(Array(Pitch(3,4)),Dur(
1,2))))))
trip8 = Dur(1,12)

-x-

56,56,
0,0,
1,1,
```

```
56,56,
0,0,
1,1,
56,56,
0,0,
1,1,
56,56,
0,0,
1,1,
60,60,60,62,64,64,62,64,65,67,60,60,60,67,67,67,64,64,64,60,60,60,67,65,64,62,60,
1,1,3,1,1,3,1,3,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,3,1,1,
4,4,16,16,4,16,16,16,16,2,12,12,12,12,12,12,12,12,12,12,12,12,16,16,16,16,2,
56,60,60,60,62,64,64,62,64,65,67,60,60,60,67,67,67,64,64,64,60,60,60,67,65,64,62,60,
1,1,1,3,1,1,3,1,3,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,3,1,1,
1,4,4,16,16,4,16,16,16,16,2,12,12,12,12,12,12,12,12,12,12,12,12,16,16,16,16,2,
56,56,60,60,60,62,64,64,62,64,65,67,60,60,60,67,67,67,64,64,64,60,60,60,67,65,64,62,60,
1,1,1,1,3,1,1,3,1,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,3,1,1,
1,1,4,4,16,16,4,16,16,16,16,2,12,12,12,12,12,12,12,12,12,12,12,12,16,16,16,16,2,
56,56,56,60,60,60,62,64,64,62,64,65,67,60,60,60,67,67,67,64,64,64,60,60,60,67,65,64,62,60,
1,1,1,1,1,3,1,1,3,1,3,1,1,1,1,1,1,1,1,1,1,1,1,1,3,1,3,1,1,
1,1,1,4,4,16,16,4,16,16,16,16,2,12,12,12,12,12,12,12,12,12,12,12,12,16,16,16,16,2,

-x-

shepard.cmaj

int compose() {
  dur d = (1,8);
  pitch[] pitches = $C ** 8;
  pitch base = $C0;
  pitch max = $C8;
  int i;
  int n = 40;

  //Initialize an array of pitches
  for(i = 0; i < 8; i = i + 1) {
    pitches[i] = base + i * 12;
  }

  //Main loop
  phrase ph = newphrase();
  for(i = 0; i < n; i = i + 1) {
    int j;
    for(j = 0; j < 8; j = j + 1) {
      pitches[j] = pitches[j] + 1;
      if(pitches[j] == max)
        pitches[j] = base;
    }

    //Put them all on top of one another
    chord ch = newchord(d);
    for(j = 0; j < 8; j = j + 1) {
      ch = ch ^ pitches[j];
    }

    ph = ph + ch;
  }

  play(ph);

}

chord newchord(dur d) {
  note n1 = ($R,d);
  return n1 ^ $R;
}

phrase newphrase() {
  chord c1 = newchord((0,1));
  chord c2 = c1;
  return c1 + c2;
}
```

```
-x-

base = Pitch(3,0)
d = Dur(1,8)
i = Int(40)
max = Pitch(3,8)
n = Int(40)
ph =
Phrase(Array(Chord(Array(Pitch(-1,4),Pitch(-1,4)),Dur(0,1)),Chord(Array(Pitch(-1,4),Pitch(-1,4)),
Dur(0,1)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(4,0),Pitch(4,1),Pitch(4,2),Pitch(4,3),Pitch(4
,
4),Pitch(4,5),Pitch(4,6),Pitch(4,7)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(5,0),Pit
ch(5,1),Pitch(5,2),Pitch(5,3),Pitch(5,4),Pitch(5,5),Pitch(5,6),Pitch(5,7)),Dur(1,8)),Chord(Array(
Pitch(-1,4),Pitch(-1,4),Pitch(6,0),Pitch(6,1),Pitch(6,2),Pitch(6,3),Pitch(6,4),Pitch(6,5),Pitch(6
,
6),Pitch(6,7)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(7,0),Pitch(7,1),Pitch(7,2),Pit
ch(7,3),Pitch(7,4),Pitch(7,5),Pitch(7,6),Pitch(7,7)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4
),Pitch(8,0),Pitch(8,1),Pitch(8,2),Pitch(8,3),Pitch(8,4),Pitch(8,5),Pitch(8,6),Pitch(8,7)),Dur(1,
8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(9,0),Pitch(9,1),Pitch(9,2),Pitch(9,3),Pitch(9,4),Pi
tch(9,5),Pitch(9,6),Pitch(9,7)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(10,0),Pitch(1
0,1),Pitch(10,2),Pitch(10,3),Pitch(10,4),Pitch(10,5),Pitch(10,6),Pitch(10,7)),Dur(1,8)),Chord(Arr
ay(Pitch(-1,4),Pitch(-1,4),Pitch(11,0),Pitch(11,1),Pitch(11,2),Pitch(11,3),Pitch(11,4),Pitch(11,5
),Pitch(11,6),Pitch(11,7)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(0,1),Pitch(0,2),Pi
tch(0,3),Pitch(0,4),Pitch(0,5),Pitch(0,6),Pitch(0,7),Pitch(0,8)),Dur(1,8)),Chord(Array(Pitch(-1,4
),Pitch(-1,4),Pitch(1,1),Pitch(1,2),Pitch(1,3),Pitch(1,4),Pitch(1,5),Pitch(1,6),Pitch(1,7),Pitch(
1,8)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(2,1),Pitch(2,2),Pitch(2,3),Pitch(2,4),P
itch(2,5),Pitch(2,6),Pitch(2,7),Pitch(2,8)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(3
,
1),Pitch(3,2),Pitch(3,3),Pitch(3,4),Pitch(3,5),Pitch(3,6),Pitch(3,7),Pitch(3,0)),Dur(1,8)),Chord(
Array(Pitch(-1,4),Pitch(-1,4),Pitch(4,1),Pitch(4,2),Pitch(4,3),Pitch(4,4),Pitch(4,5),Pitch(4,6),P
itch(4,7),Pitch(4,0)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(5,1),Pitch(5,2),Pitch(5
,
3),Pitch(5,4),Pitch(5,5),Pitch(5,6),Pitch(5,7),Pitch(5,0)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitc
h(-1,4),Pitch(6,1),Pitch(6,2),Pitch(6,3),Pitch(6,4),Pitch(6,5),Pitch(6,6),Pitch(6,7),Pitch(6,0)),
Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(7,1),Pitch(7,2),Pitch(7,3),Pitch(7,4),Pitch(7
,
5),Pitch(7,6),Pitch(7,7),Pitch(7,0)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(8,1),Pit
ch(8,2),Pitch(8,3),Pitch(8,4),Pitch(8,5),Pitch(8,6),Pitch(8,7),Pitch(8,0)),Dur(1,8)),Chord(Array(
Pitch(-1,4),Pitch(-1,4),Pitch(9,1),Pitch(9,2),Pitch(9,3),Pitch(9,4),Pitch(9,5),Pitch(9,6),Pitch(9
,
7),Pitch(9,0)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(10,1),Pitch(10,2),Pitch(10,3),
Pitch(10,4),Pitch(10,5),Pitch(10,6),Pitch(10,7),Pitch(10,0)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pi
tch(-1,4),Pitch(11,1),Pitch(11,2),Pitch(11,3),Pitch(11,4),Pitch(11,5),Pitch(11,6),Pitch(11,7),Pit
ch(11,0)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(0,2),Pitch(0,3),Pitch(0,4),Pitch(0,
5),Pitch(0,6),Pitch(0,7),Pitch(0,8),Pitch(0,1)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pit
ch(1,2),Pitch(1,3),Pitch(1,4),Pitch(1,5),Pitch(1,6),Pitch(1,7),Pitch(1,8),Pitch(1,1)),Dur(1,8)),C
hord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(2,2),Pitch(2,3),Pitch(2,4),Pitch(2,5),Pitch(2,6),Pitch(2
,
7),Pitch(2,8),Pitch(2,1)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(3,2),Pitch(3,3),Pit
ch(3,4),Pitch(3,5),Pitch(3,6),Pitch(3,7),Pitch(3,0),Pitch(3,1)),Dur(1,8)),Chord(Array(Pitch(-1,4)
,Pitch(-1,4),Pitch(4,2),Pitch(4,3),Pitch(4,4),Pitch(4,5),Pitch(4,6),Pitch(4,7),Pitch(4,0),Pitch(4
,
1)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(5,2),Pitch(5,3),Pitch(5,4),Pitch(5,5),Pit
ch(5,6),Pitch(5,7),Pitch(5,0),Pitch(5,1)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(6,2
),Pitch(6,3),Pitch(6,4),Pitch(6,5),Pitch(6,6),Pitch(6,7),Pitch(6,0),Pitch(6,1)),Dur(1,8)),Chord(A
rray(Pitch(-1,4),Pitch(-1,4),Pitch(7,2),Pitch(7,3),Pitch(7,4),Pitch(7,5),Pitch(7,6),Pitch(7,7),Pi
tch(7,0),Pitch(7,1)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(8,2),Pitch(8,3),Pitch(8,
4),Pitch(8,5),Pitch(8,6),Pitch(8,7),Pitch(8,0),Pitch(8,1)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitc
h(-1,4),Pitch(9,2),Pitch(9,3),Pitch(9,4),Pitch(9,5),Pitch(9,6),Pitch(9,7),Pitch(9,0),Pitch(9,1)),
Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(10,2),Pitch(10,3),Pitch(10,4),Pitch(10,5),Pit
ch(10,6),Pitch(10,7),Pitch(10,0),Pitch(10,1)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch
(11,2),Pitch(11,3),Pitch(11,4),Pitch(11,5),Pitch(11,6),Pitch(11,7),Pitch(11,0),Pitch(11,1)),Dur(1
,
8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(0,3),Pitch(0,4),Pitch(0,5),Pitch(0,6),Pitch(0,7),Pi
tch(0,8),Pitch(0,1),Pitch(0,2)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(1,3),Pitch(1,
4),Pitch(1,5),Pitch(1,6),Pitch(1,7),Pitch(1,8),Pitch(1,1),Pitch(1,2)),Dur(1,8)),Chord(Array(Pitch
(-1,4),Pitch(-1,4),Pitch(2,3),Pitch(2,4),Pitch(2,5),Pitch(2,6),Pitch(2,7),Pitch(2,8),Pitch(2,1),P
itch(2,2)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(3,3),Pitch(3,4),Pitch(3,5),Pitch(3
,
6),Pitch(3,7),Pitch(3,0),Pitch(3,1),Pitch(3,2)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pit
ch(4,3),Pitch(4,4),Pitch(4,5),Pitch(4,6),Pitch(4,7),Pitch(4,0),Pitch(4,1),Pitch(4,2)),Dur(1,8)),C
hord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(5,3),Pitch(5,4),Pitch(5,5),Pitch(5,6),Pitch(5,7),Pitch(5
,
```

```
0),Pitch(5,1),Pitch(5,2)),Dur(1,8)),Chord(Array(Pitch(-1,4),Pitch(-1,4),Pitch(6,3),Pitch(6,4),Pit
ch(6,5),Pitch(6,6),Pitch(6,7),Pitch(6,0),Pitch(6,1),Pitch(6,2)),Dur(1,8)),Chord(Array(Pitch(-1,4)
,Pitch(-1,4),Pitch(7,3),Pitch(7,4),Pitch(7,5),Pitch(7,6),Pitch(7,7),Pitch(7,0),Pitch(7,1),Pitch(7
,2)),Dur(1,8))))
pitches =
Array(Pitch(7,3),Pitch(7,4),Pitch(7,5),Pitch(7,6),Pitch(7,7),Pitch(7,0),Pitch(7,1),Pitch(7,2))

-x-

-1,-1,97,56,99,56,101,56,103,56,105,56,107,56,13,56,15,56,17,56,19,56,21,56,23,56,25,56,27,56,29,
56,31,56,33,56,35,56,37,56,39,56,
0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
-1,-1,85,56,87,56,89,56,91,56,93,56,95,56,97,56,99,56,101,56,103,56,105,56,107,56,13,56,15,56,17,
56,19,56,21,56,23,56,25,56,27,56,
0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
-1,-1,73,14,75,16,77,18,79,20,81,22,83,24,85,26,87,28,89,30,91,32,93,34,95,36,97,38,99,40,101,42,
103,44,105,46,107,48,13,50,15,52,
0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
-1,-1,61,26,63,28,65,30,67,32,69,34,71,36,73,38,75,40,77,42,79,44,81,46,83,48,85,50,87,52,89,54,9
1,56,93,58,95,60,97,62,99,64,
0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
-1,-1,49,38,51,40,53,42,55,44,57,46,59,48,61,50,63,52,65,54,67,56,69,58,71,60,73,62,75,64,77,66,7
9,68,81,70,83,72,85,74,87,76,
0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
-1,-1,37,50,39,52,41,54,43,56,45,58,47,60,49,62,51,64,53,66,55,68,57,70,59,72,61,74,63,76,65,78,6
7,80,69,82,71,84,73,86,75,88,
0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
-1,-1,25,62,27,64,29,66,31,68,33,70,35,72,37,74,39,76,41,78,43,80,45,82,47,84,49,86,51,88,53,90,5
5,92,57,94,59,96,61,98,63,100,
0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
-1,-1,13,74,15,76,17,78,19,80,21,82,23,84,25,86,27,88,29,90,31,92,33,94,35,96,37,98,39,100,41,102
,43,104,45,106,47,12,49,14,51,16,
0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
56,56,56,86,56,88,56,90,56,92,56,94,56,96,56,98,56,100,56,102,56,104,56,106,56,12,56,14,56,16,56,
18,56,20,56,22,56,24,56,26,56,28,
0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
56,56,56,98,56,100,56,102,56,104,56,106,56,12,56,14,56,16,56,18,56,20,56,22,56,24,56,26,56,28,56,
30,56,32,56,34,56,36,56,38,56,40,
0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
```

6.3 Test Suites

6.3.1 Motivation

Test cases were chosen to test individual features of the language (such as variable declaration, operator functionality, control structures, etc.) as independently of each other as possible. This makes it easier to debug our compiler and feature implementation. Test cases were also written to be as thorough as possible. For example, for operators, the operator is tested using all the different possible type operand combinations.

6.3.2 Automation

Testing is automated using the test.sh file and the test suite can be run by executing the command ./test.sh on the command line.

Laura was our Lead Tester and wrote the test suite, automated test script, and expected output files. Most of the individual test cases testing different components and features of the language were written by the person who implemented the features.

7 Lessons Learned

7.1 Andrew O'Reilly

If you have six tasks and three people, and you tell each of them to pick two and do them, very little will get done. The most invasive and dictatorial management styles are probably the most effective, to the extent that they do not upset everyone else. If your personality renders you incapable of this, or if your management style relies on sticks as well as carrots, you should not be the manager of this project, as the sticks available to you in this context will be limited (as will the carrots). Communication is of the utmost importance, and if there are team members who do not communicate effectively be sure to communicate this to them. If you set your own internal deadlines aside from those given by the professor, write them down or post them somewhere.

I have further learned that functional programming is all about answering questions and elucidation of meaning in a programming context. A functional compiler is constantly asking you what is returned by some set of code, what that result means, and whether it makes sense. Try to keep this in mind if you are to write functional code.

7.2 Stephanie Huang

Learning OCaml was a challenging, but rewarding experience. One of the key takeaways I got from the overall experience of writing in OCaml and also designing a programming language is the idea that you should try to do more with less. OCaml is all about doing a lot with a little bit of code, and similarly, our programming language is trying to do a lot with a few lines of code.

As for working as a team, and projects in general: communication is key. It's important to always keep in touch, schedule regular meeting times and checkins, communicate your own ideas and what you've been working on. A lot of project work is also taking the initiative to do something, especially when working with peers when there isn't as much of a topdown structure as there might be in some real world working environments. Try to set milestones and plan a timeline in advance, maybe even from day one.

7.3 Jonathan Sun

Keep things simple. Have a written long term game plan from the start. Don't be married to any initial idea, learn to let go, allow yourself to be convinced of new ideas. Make attempts at documentation. Don't touch working code. Always ask for help right away. Any embarrassment that holds you back will damage the group's progress.

7.4 Laura Tang

For many students, including myself, this class is one of the first where you are required to complete a semester long group project with a team of other students. On top of that, the project requires you to get used to programming in OCaml, employing the functional programming paradigm rather than imperative programming that most are more familiar with. This lack of global view may cause analysis paralysis, so to resolve that, it is best to have an organized leader and to break down the tasks early after a thorough brainstorm with your members. Personally assigned tasks made of smaller chunks also are best, as they pre-commit members to have ownership of their own small goal.

I learned that it's best to remain transparent: any changes that are made to the design should be communicated to all members, whether it is through the group chat, Github updates, a workboard, etc. Any work that you do on the project, work that may take longer than expected, and bugs you found should all be reported. In addition, I found that it?s important that each member keeps up to date on those changes, because the implementation of the language features will definitely be modified along the way.

8 Appendix

8.1 Source Code

ast.ml

```
1 (*
2 CMajor AST
3 by PLT Sandwich
4 Andrew OReilly, Stephanie Huang
5 *)
6
7 (* All operators. TODO: make sure this is complete *)
```

```
type operator =   Add | Sub | Mul | Div | Layer | Arrcat | Rep
                | Eq | Neq | Gt | Gte | Lt | Lte

(* Variable declaration. See parser for initialization *)
type vdecl = string * string

(* Literal types *)
type literal =
    Intlit of int                (* Integers - 42 *)
  | Pitchlit of int * int        (* Pitches - $Cb7 *)
  | Tuple of expr * expr         (* Tuples - notes, durations *)

and expr =
    Binop of expr * operator * expr (* Binary operations *)
  | Noexpr                          (* Empty expression *)
  | Lit of literal                  (* Literals are expressions *)
  | Asn of string * expr            (* Variable assignment *)
  | Id of string                    (* Identifiers *)
  | Call of string * expr list      (* Function call *)
  | Arr of expr list                (* Arrays *)
  | Arrget of string * expr         (* Array reference *)
  | Arrmod of string * expr * expr  (* Array modification *)

(* Statements *)
type stmt =
    Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | Vdecl of string * string
  | VdeclAsn of stmt * expr

(* Function declaration. Args list is now a vdecl list *)
type func_decl = {
  ftype: string;
  fname: string;
  formals: vdecl list;
  locals : string list;
  body: stmt list;
}

(* type program = stmt *)
type program = vdecl list * func_decl list
```

cmajor.ml

```
(*
cmajor.ml
Main executable for C-Major
By PLT Sandwich
Andrew OReilly, Stephanie Huang
*)

open Ast
open Sast
open Compile
open Semantics
```

```
open Printf

let _ =
  let chan = if Array.length Sys.argv = 2
  then
    try
      Some(open_in Sys.argv.(1))
    with Sys_error(s) ->
      print_endline ("Error: " ^ s); None
  else (
    Some(stdin)
  )
  in match chan with
      Some(channel) ->
        let lexbuf = Lexing.from_channel channel in
        let prog = Parser.program Scanner.token lexbuf in
        Execute.execute_prog prog
    | _ -> exit 1;
```

compile.ml

```
(*
compile.ml
Main compiler for the CMajor programming language
by PLT Sandwich
Andrew OReilly, Stephanie Huang
*)

open Ast
open Sast
open Semantics
open Printf

exception Invalid_play of string
exception Illegal_operation of string
exception Type_error of string
exception Duplicate_name of string

exception Not_implemented of string

(* Symbol map *)
module NameMap = Map.Make(String)

(* Environment: symbol tables for functions, global, local vars *)
type global_environment = {
  variables : (c_type * s_type) NameMap.t;
  functions : func_decl NameMap.t
}

type symbol_table = {
  parent : symbol_table option;
  variables : (c_type * s_type) NameMap.t
}

type env = {
    glob_env : global_environment;
    scope : symbol_table;
    return_type : s_type;
```

```
38      return_val : c_type;
39  }
40
41  type composition = {
42    dur1 : int list;
43    dur2 : int list;
44    pitches : int array list
45  }
46
47  (* extract value from c_type * s_type tuple *)
48  let get_value (v, t) = match v with
49      | None -> raise Not_found
50      | _ -> v
51
52  (* execute play by writing pitches and durations to composition *)
53  let play s =
54    (* next is a phrase *)
55    let handle_phrase next l = match next with Phrase(chords) ->
56      let comp = { dur1 = []; dur2 = []; pitches = [Array.make (Array.length chords) (-1)
       ]}
57      in let rec match_length llen alen l =
58        if llen >= alen then l else
59        match_length (llen + 1) alen ((Array.make (Array.length chords) (-1)) :: l)
60      in
61      (* handles the pitches *)
62      let (na, new_pitches) = (
63        Array.fold_left (fun (i, comp_pitches) chord ->
64          match chord with Chord(p,d) ->
65            let new_pitches =
66              List.rev (match_length (List.length comp_pitches) (Array.length p)
       comp_pitches)
67            in Array.iteri (fun pidx pitch ->
68              (List.nth new_pitches pidx).(i) <- (pitch_to_int pitch)
69            ) p; (i + 1, new_pitches)
70          | _ -> raise (Type_mismatch ("Error in play"))
71        ) (0, comp.pitches) chords)
72      (* handles the durations *)
73      in (Array.fold_right (fun chord comp -> match chord with Chord(p,d) ->
74        (match d with Dur(d1, d2) ->
75          {
76            dur1 = d1 :: comp.dur1;
77            dur2 = d2 :: comp.dur2;
78            pitches = new_pitches;
79          }
80        | _ -> raise (Type_mismatch ("Error in play")))
81      | _ -> raise (Type_mismatch ("Error in play"))
82      ) chords comp
83    ) :: l
84    | _ -> raise (Type_mismatch ("Error in play"))
85    in match s with Score(phrases) ->
86      Array.fold_right handle_phrase phrases []
87    | _ -> raise (Type_mismatch ("Error in play"))
88
89  let csv_ints (listarg : int list) =
90    List.fold_right
91     (fun next str -> (string_of_int next) ^ "," ^ str)
92     listarg ""
93
```

```
94  (* Get the value of a variable from a symbol_table *)
95  let rec find_symtab_var symtab name =
96    try
97      NameMap.find name symtab.variables
98    with Not_found ->
99      match symtab.parent with
100         Some(parent) -> find_symtab_var parent name
101       | _ -> raise Not_found
102
103 (* Get the value of a variable from an environment *)
104 let find_var env name =
105   try
106     find_symtab_var env.scope name
107   with Not_found -> try
108     find_symtab_var {
109       parent = None;
110       variables = env.glob_env.variables
111     } name
112   with Not_found -> raise Not_found
113
114 (* Set the value of a variable within a symbol_table *)
115 let rec update_var symtab name newval (t : s_type) =
116   try
117     (* update local scope *)
118     let (oldval, typ) = NameMap.find name symtab.variables  in
119     if t <> typ then raise (Type_mismatch(name))
120     else
121     (* check array typing *)
122       if t = SArray then if get_arr_type newval <> get_arr_type oldval
123         then raise (Type_mismatch(name)) (*else ()*)
124       else (); {
125         parent = symtab.parent;
126         variables = NameMap.add name (newval, t) symtab.variables
127       }
128   with Not_found ->
129     match symtab.parent with
130         Some(psymtab) ->
131           let newparent = update_var psymtab name newval t in
132           {
133             parent = Some(newparent);
134             variables = symtab.variables
135           }
136       | _ -> raise Not_found
137
138 (* Set the value of a variable within an environment *)
139 let update_symtab env name newval (t : s_type) =
140   try
141     let new_symtab = update_var env.scope name newval t in
142     newval, {
143       glob_env = env.glob_env;
144       scope = new_symtab;
145       return_type = env.return_type;
146       return_val = env.return_val
147     }
148   with Not_found -> (
149     try
150       let new_glob_scope = update_var {
151         parent = None;
```

```
152         variables = env.glob_env.variables
153       } name newval t in
154       newval, {
155         glob_env = {
156           variables = new_glob_scope.variables;
157           functions = env.glob_env.functions
158         };
159         scope = env.scope;
160         return_type = env.return_type;
161         return_val = env.return_val
162       }
163     with Not_found -> raise Not_found
164   )
165   | Match_failure(s,l,c) -> raise (Type_error(name ^ c_type_str(newval)))
166
167
168 let rec translate block env = (
169   (* translate all expressions to cmaj type *)
170     let rec expr expenv = function
171         Id(s) -> (
172           try
173             let (c,s) = find_var expenv s in
174             (c, expenv)
175           with Not_found -> raise (Missing_variable ("Error: "^"\""^s^"\""^" not
    defined!"))
176        )
177     | Asn(s, e) -> (
178       try
179         let (c, env1) = (expr expenv e) in
180         (* cast array to phrase or score as necessary *)
181         let cast_check = function
182             SPhrase -> (match c with Array(SChord, dat) -> Phrase(dat) | _ -> c)
183           | SScore -> (match c with Array(SPhrase, dat) -> Score(dat) | _ -> c)
184           | _ -> c
185         in let c = cast_check (snd (find_var env1 s)) in
186         (* assign value in environment *)
187         update_symtab env1 s c (c_to_s_type c)
188       with Not_found -> raise (Missing_variable ("Error: "^"\""^s^"\""^" not defined!")
    )
189        )
190     | Arr(e) -> (
191       let arr_type = c_to_s_type (fst (expr expenv (List.hd e))) in
192       let arr_check elem =
193         let c_elem = fst (expr expenv elem) in
194         if (is_valid_elem c_elem arr_type) then c_elem
195         else raise (Type_error("Error: unexpected type encountered"))
196       in let arr_data = Array.of_list (List.map arr_check e) in
197       Array(arr_type, arr_data), expenv
198     )
199     | Arrget(s, e) -> (
200       try
201         let (arr, env1) = expr expenv (Id(s)) in
202         let index = match (fst (expr env1 e)) with
203           Int(i) -> i
204         | _ -> raise (Type_error ("Error: index value is not an integer"))
205         in let arr_data = match arr with
206           Array(typ, dat) -> dat
207         | _ -> raise (Type_error ("Error: \""^s^"\" is not an array"))
```

```
208           in arr_data.(index), env1
209         with Invalid_argument x -> raise (Invalid_argument ("index out of bounds"))
210       )
211       | Arrmod(s, i, e) -> (
212         try
213           let (arr, env1) = expr expenv (Id(s)) in
214           let index = match (fst (expr env1 i)) with
215             Int(idx) -> idx
216           | _ -> raise (Type_error ("Error: index value is not an integer"))
217           in let arr_data = match arr with
218             Array(typ, dat) -> dat
219           | _ -> raise (Type_error ("Error: \""^s^"\" is not an array"))
220           in let newval = fst (expr env1 e)
221           in if is_valid_elem newval (get_arr_type arr)
222             then arr_data.(index) <- newval
223           else raise (Type_error ("Error: unexpected type encountered"));
224           newval, env1
225         with Invalid_argument x -> raise (Invalid_argument ("index out of bounds"))
226       )
227       | Lit(x) -> (
228         match x with
229             Intlit(x) -> Int(x), expenv
230           | Pitchlit(l,o) -> Pitch(l, o), expenv
231           | Tuple(x, y) -> (
232             let (ex, env1) = expr expenv x  in
233               let (ey, env2) = expr env1 y  in
234               match ex, ey with
235                 Int(a), Int(b) -> Dur(a, b), env2
236               | Pitch(l, o), Dur(a, b) -> Note(Pitch(l,o), Dur(a,b)), env2
237               | Array(SPitch, d), Dur(a, b) -> Chord(d, Dur(a,b)), env2
238               | _ -> raise (Type_error("Invalid tuple"))
239             )
240         )
241         | Call(name, elist) -> (
242             let argc, argv, callenv = List.fold_left
243               (
244                 fun (c,v,envb) next ->
245                 let lit, enva = expr envb next in
246                   c + 1, lit :: v, enva
247               )
248               (0, [], expenv) elist
249             in
250             match name with
251             "play" -> (
252               match argv with
253               [_ as s] -> (
254                 let master_score = match s with
255                   Score(phrases) -> Score(phrases)
256                 | Phrase(chords) as p -> Score([|p|])
257                 | Chord(p,d) as c -> Score([|Phrase([|c|])|])
258                 | Note(p,d) -> Score([|Phrase([|Chord([|p|],d)|])|])
259                 | _ -> raise (Invalid_play("Invalid call to play"))
260                 in let comp_list = play master_score in
261                 let ofile = open_out "out.csv" in
262                 List.iter (fun comp ->
263                   let (dur1str, dur2str) =
264                     (csv_ints comp.dur1, csv_ints comp.dur2)
265                   in List.iter (fun parray ->
```

```
266                   let pitchstr = csv_ints (Array.to_list parray) in
267                   fprintf ofile "%s\n%s\n%s\n" pitchstr dur1str dur2str;
268                 ) comp.pitches
269               ) comp_list;
270               close_out ofile;
271               Int(1), expenv
272             )
273             | _ -> raise (Invalid_play("Invalid call to play"))
274           )
275           | _ ->
276               try
277                 let fxn = NameMap.find name callenv.glob_env.functions in
278                 let funenv = exec_fun callenv fxn (List.rev argv) in
279                 funenv.return_val, {
280                   glob_env = funenv.glob_env;
281                   scope = callenv.scope;
282                   return_type = callenv.return_type;
283                   return_val = callenv.return_val
284                 }
285               with Not_found -> raise (Missing_function("Unknown function " ^ name))
286           (*raise (Not_implemented("Custom functions"))*)
287       )
288     | Binop (e1, op, e2) -> (
289       let (lit1, env1) = expr expenv e1 in
290       let (lit2, binenv) = expr env1 e2 in
291       match op with
292         Add -> (
293           match (lit1, lit2) with
294             Int(x), Int(y) -> Int(x + y), binenv
295           | (Pitch(l,o) as p), (Int(y) as i)
296           | (Int(y) as i), (Pitch(l,o) as p) -> raise_pitch i p, binenv
297           | Dur(x,y), Int(z)
298           | Int(z), Dur(x,y) -> dur_add (Dur(z,1)) (Dur(x,y)), binenv
299           | Dur(x,y), Dur(z,w) -> dur_add (Dur(x,y)) (Dur(z,w)), binenv
300           (* as concatenation *)
301           | Note(p1,d1), Note(p2,d2) -> Phrase([| Chord([|p1|],d1); Chord([|p2|],d2)
      |]), binenv
302           | Note(p,d), (Chord(p2,d2) as c) -> Phrase([| Chord([|p|], d); c |]), binenv
303           | (Chord(p2,d2) as c), Note(p,d) -> Phrase([| c; Chord([|p|], d) |]), binenv
304           | Note(p,d), Phrase(c) -> Phrase(Array.append [| Chord([|p|],d) |] c), binenv
305           | Phrase(c), Note(p,d) -> Phrase(Array.append c [| Chord([|p|],d) |]), binenv
306           | Chord(p1,d1) as c1, (Chord(p2,d2) as c2) -> Phrase([| c1; c2 |]), binenv
307           | Phrase(c), (Chord(p,d) as ch) -> Phrase(Array.append c [|ch|]), binenv
308           | Chord(p,d) as ch, Phrase(c) -> Phrase(Array.append [|ch|] c), binenv
309           | Phrase(c1), Phrase(c2) -> Phrase(Array.append c1 c2), binenv
310           | _ as f, (_ as s) ->
311             raise (Illegal_operation((c_type_str f) ^ "+" ^ (c_type_str s)))
312       )
313       | Sub ->
314           let lit = (
315             match lit1, lit2 with
316                 Int(x), Int(y) -> Int(x-y)
317               | Pitch(letr,oct), Int(x) ->
318                   let rcomp = letr - x in
319                   if rcomp < 0 then
320                     let posval = rcomp * -1 in
321                     let mdls = posval mod 12 in
322                     let submod = 12 - mdls in
```

```
323                         let oreduce = posval/12 in
324                         let oret = oct - (oreduce+1) in
325                         if oret < 0
326                         then Pitch(0,0)
327                         else Pitch(submod,oret)
328                       else Pitch(rcomp, oct)
329                   | Dur(num,den) as d, Int(x) ->
330                       dur_sub_abs (d, Dur(x,1))
331                   | Pitch(l1,o1) as p1, (Pitch(l2,o2) as p2) ->
332                       let i1 = pitch_to_int p1 in
333                       let i2 = pitch_to_int p2 in
334                       Int(i1 - i2)
335                   | Chord(p_array,d), (Pitch(l,o) as p) ->
336                       let pitch_list = Array.fold_left
337                         (fun plist next ->
338                           if next = p then plist else next :: plist
339                         ) [] p_array
340                       in let pitches = Array.of_list pitch_list
341                       in Chord(pitches,d)
342                   | Dur(_,_) as d1, (Dur(_,_) as d2) ->
343                       dur_sub_abs (d1,d2)
344                   | Note(p,d1), (Dur(_,_) as d2) ->
345                       Note(p, (dur_sub_abs (d1,d2)))
346                   | Chord(p,d1), (Dur(_,_) as d2) ->
347                       Chord(p, (dur_sub_abs (d1,d2)))
348                   | _ as f, (_ as s) ->
349                       raise (Illegal_operation((c_type_str f) ^ "-" ^ (c_type_str s)))
350               ) in lit, binenv
351           | Mul ->
352               let lit = (
353                 match lit1, lit2 with
354                     Int(x), Int(y) -> Int(x*y)
355                   | Dur(n,d), Int(x)
356                   | Int(x), Dur(n,d) ->
357                       let rn = n*x in
358                       let g = gcd rn d in
359                       Dur(rn/g,d/g)
360                   | Dur(n1,d1), Dur(n2,d2) ->
361                       let n = n1 * n2 in
362                       let d = d1 * d2 in
363                       let g = gcd n d in
364                       Dur(n/g,d/g)
365                   | _ as f, (_ as s) ->
366                       raise (Illegal_operation((c_type_str f) ^ "*" ^ (c_type_str s)))
367               ) in lit, binenv
368           | Div -> (
369               match (lit1, lit2) with
370                     Int(x), Int(y)            -> Dur(x, y), binenv
                    (* int / int *)
371                   | Dur(n,d), Int(i)          -> dur_divide (Dur(n,d), Int(i)), binenv
                    (* dur / int *)
372                   | Note(p, d), Int(i)        -> Note( p, dur_divide (d, Int(i)) ), binenv
                    (* note / int *)
373                   | Chord(p, d), Int(i)       -> Chord( p, dur_divide (d, Int(i)) ), binenv
                    (* chord / int *)
374                   | Int(x), Dur(n,d)          -> dur_divide (Int(x), Dur(n,d)), binenv
                    (* int / dur *)
375                   | Dur(n1,d1), Dur(n2,d2)    -> dur_divide (Dur(n1,d1), Dur(n2,d2)),
```

```
    binenv          (* dur / dur *)
376             | Note(p, dur), Dur(n,d)    -> Note( p, dur_divide (dur, Dur(n,d)) ),
    binenv       (* note / dur *)
377             | Chord(p, chdur), Dur(n,d) -> Chord( p, dur_divide (chdur, Dur(n,d)) ),
    binenv    (* chord / dur *)
378             | Dur(n,d), Note(p, dur)    -> Note(p, dur_divide (Dur(n,d), dur)),
    binenv          (* dur / note *)
379             | Dur(n,d), Chord(p, chdur) -> Chord( p, dur_divide (chdur, Dur(n,d)) ),
    binenv    (* dur / chord *)
380             | _ as f, (_ as s)
381               -> raise (Illegal_operation((c_type_str f) ^ "/" ^ (c_type_str s)))
382         )
383       | Layer -> (
384         match (lit1, lit2) with
385           (Dur(n,d1) as d), (Pitch(l,o) as p)
386         | (Pitch(l,o) as p), (Dur(n,d1) as d) -> Note(p, d), binenv
387         (* TODO: array of pitches and array of durations (* Done? *)*)
388         | Array(SPitch, p), Array(SDur, d)
389         | Array(SDur, d), Array(SPitch, p) -> (
390           if (Array.length p) != (Array.length d) then
391             raise (Illegal_operation ("Error: pitch[] and dur[] must be same length")
    )
392           else
393             let ph = Array.mapi (fun i e -> Chord([|e|], d.(i))) p in
394             Phrase(ph), binenv
395         )
396         | (Pitch(l,o) as pitch), Note(p,d)
397         | Note(p,d), (Pitch(l,o) as pitch) ->
398             Chord([| p; pitch |], d), binenv
399         | (Pitch(l,o) as pitch), Chord(p,d)
400         | Chord(p,d), (Pitch(l,o) as pitch) ->
401             Chord(Array.append p [| pitch |], d), binenv
402         | Note(p1,d1), (Chord(p_array,d2) as c) ->
403             Score([| Phrase([|Chord([|p1|],d1)|]); Phrase([|c|]) |]), binenv
404         | (Chord(p_array,d2) as c), Note(p1,d1) ->
405             Score([| Phrase([|c|]); Phrase([|Chord([|p1|],d1)|]) |]), binenv
406         | (Chord(p1,d1) as c1), (Chord(p2,d2) as c2) ->
407             Score([| Phrase([|c1|]); Phrase([|c2|]) |]), binenv
408         | (Chord(p,d) as c), (Phrase(s) as ph) ->
409             Score([| Phrase([|c|]); ph |]), binenv
410         | (Phrase(s) as ph), (Chord(p,d) as c) ->
411             Score([| ph; Phrase([|c|]) |]), binenv
412         | (Phrase(c1) as p1), (Phrase(c2) as p2) ->
413             Score([| p1; p2 |]), binenv
414         | (Phrase(c) as p), Score(ph)
415         | Score(ph), (Phrase(c) as p) -> Score(Array.append ph [|p|]), binenv
416         | _ as f, (_ as s) ->
417             raise (Illegal_operation((c_type_str f) ^ "^" ^ (c_type_str s)))
418       )
419       | Arrcat -> (
420         match (lit1, lit2) with
421           Array(t1, d1), Array(t2, d2) ->
422             if t1 = t2 then Array(t1, Array.append d1 d2), binenv
423             else raise (Illegal_operation("Error: cannot concatenate arrays of two
    types"))
424         | Array(t, d), x ->
425             if c_to_s_type x = t then Array(t, Array.append d [|x|]), binenv
426             else raise (Illegal_operation("Error: type mismatch"))
```

```
427             | x, Array(t,d) ->
428                 if c_to_s_type x = t then Array(t, Array.append [|x|] d), binenv
429                 else raise (Illegal_operation("Error: type mismatch"))
430             | x, y ->
431                 if c_to_s_type x = c_to_s_type y then
432                   Array(c_to_s_type x, [| x; y |]), binenv
433                 else raise(Illegal_operation("Error: type mismatch"))
434             )
435           | Rep ->
436               let lit = (
437                 match lit1, lit2 with
438                     _ as nonint, Int(x)
439                   | Int(x), (_ as nonint) ->
440                       let arr_type = c_to_s_type nonint
441                       in let arr_data = Array.make x nonint
442                       in Array(arr_type, arr_data)
443                   | _ as f, (_ as s) ->
444                       raise (Illegal_operation(
445                         "** must be used with at least 1 int, here used with "
446                         ^ (c_type_str f) ^ " and " ^ (c_type_str s)
447                       ))
448               ) in lit, binenv
449           | Eq -> ( match lit1, lit2 with
450                 Int(x), Int(y) ->
451                   let r = if x = y then 1 else 0 in Int(r),binenv
452               | Dur(n1,d1), Dur(n2,d2) ->
453                   let r = if n1 = n2 && d1 = d2 then 1 else 0
454                   in Int(r),binenv
455               | Pitch(l1,o1), Pitch(l2,o2) ->
456                   let r = if l1 = l2
457                     && o1 = o2
458                   then 1 else 0
459                   in Int(r),binenv
460               | Note(p1,d1), Note(p2,d2) ->
461                   let r = if p1 = p2 && d1 = d2 then 1 else 0
462                   in Int(r),binenv
463               | _ as f, (_ as s) ->
464                   raise (Illegal_operation((c_type_str f) ^ "==" ^ (c_type_str s)))
465             )
466           | Neq -> ( match lit1, lit2 with
467                 Int(x), Int(y) ->
468                   let r = if x = y then 0 else 1 in Int(r),binenv
469               | Dur(n1,d1), Dur(n2,d2) ->
470                   let r = if n1 = n2 && d1 = d2 then 0 else 1
471                   in Int(r),binenv
472               | Pitch(l1,o1), Pitch(l2,o2) ->
473                   let r = if l1 = l2
474                     && o1 = o2
475                   then 0 else 1
476                   in Int(r),binenv
477               | Note(p1,d1), Note(p2,d2) ->
478                   let r = if p1 = p2 && d1 = d2 then 0 else 1
479                   in Int(r),binenv
480               | _ as f, (_ as s) ->
481                   raise (Illegal_operation((c_type_str f) ^ "!=" ^ (c_type_str s)))
482             )
483           | Gt -> ( match lit1, lit2 with
484                 Int(x), Int(y) ->
```

```
485                 let r = if x > y then 1 else 0 in Int(r),binenv
486               | Dur(n1,d1) as dur1, (Dur(n2,d2) as dur2) ->
487                 let ctype = dur_sub (dur1,dur2) in
488                 ( match ctype with
489                     Dur(sn,sd) -> let r = if sn > 0 then 1 else 0 in
490                       Int(r),binenv
491                   | _ -> raise (Illegal_operation("Problem with dur_sub"))
492                 )
493               | Pitch(p1,o1) as pitch1, (Pitch(p2,o2) as pitch2) ->
494                 let i1 = pitch_to_int pitch1 in
495                 let i2 = pitch_to_int pitch2 in
496                 let r = if i1 > i2 then 1 else 0 in
497                 Int(r),binenv
498               | _ as f, (_ as s) ->
499                 raise (Illegal_operation((c_type_str f) ^ ">" ^ (c_type_str s)))
500           )
501         | Gte -> ( match lit1, lit2 with
502               Int(x), Int(y) ->
503                 let r = if x >= y then 1 else 0 in Int(r),binenv
504               | Dur(n1,d1) as dur1, (Dur(n2,d2) as dur2) ->
505                 let ctype = dur_sub (dur1,dur2) in
506                 ( match ctype with
507                     Dur(sn,sd) -> let r = if sn >= 0 then 1 else 0 in
508                       Int(r),binenv
509                   | _ -> raise (Illegal_operation("Problem with dur_sub"))
510                 )
511               | Pitch(p1,o1) as pitch1, (Pitch(p2,o2) as pitch2) ->
512                 let i1 = pitch_to_int pitch1 in
513                 let i2 = pitch_to_int pitch2 in
514                 let r = if i1 >= i2 then 1 else 0 in
515                 Int(r),binenv
516               | _ as f, (_ as s) ->
517                 raise (Illegal_operation((c_type_str f) ^ ">=" ^ (c_type_str s)))
518           )
519         | Lt -> ( match lit1, lit2 with
520               Int(x), Int(y) ->
521                 let r = if x < y then 1 else 0 in Int(r),binenv
522               | Dur(n1,d1) as dur1, (Dur(n2,d2) as dur2) ->
523                 let ctype = dur_sub (dur1,dur2) in
524                 ( match ctype with
525                     Dur(sn,sd) -> let r = if sn < 0 then 1 else 0 in
526                       Int(r),binenv
527                   | _ -> raise (Illegal_operation("Problem with dur_sub"))
528                 )
529               | Pitch(p1,o1) as pitch1, (Pitch(p2,o2) as pitch2) ->
530                 let i1 = pitch_to_int pitch1 in
531                 let i2 = pitch_to_int pitch2 in
532                 let r = if i1 < i2 then 1 else 0 in
533                 Int(r),binenv
534               | _ as f, (_ as s) ->
535                 raise (Illegal_operation((c_type_str f) ^ "<" ^ (c_type_str s)))
536           )
537         | Lte -> ( match lit1, lit2 with
538               Int(x), Int(y) ->
539                 let r = if x <= y then 1 else 0 in Int(r),binenv
540               | Dur(n1,d1) as dur1, (Dur(n2,d2) as dur2) ->
541                 let ctype = dur_sub (dur1,dur2) in
542                 ( match ctype with
```

```
                    Dur(sn,sd) -> let r = if sn <= 0 then 1 else 0 in
                      Int(r),binenv
                  | _ -> raise (Illegal_operation("Problem with dur_sub"))
                )
            | Pitch(p1,o1) as pitch1, (Pitch(p2,o2) as pitch2) ->
                let i1 = pitch_to_int pitch1 in
                let i2 = pitch_to_int pitch2 in
                let r = if i1 <= i2 then 1 else 0 in
                Int(r),binenv
            | _ as f, (_ as s) ->
                raise (Illegal_operation((c_type_str f) ^ "<=" ^ (c_type_str s)))
          )
    )
    | Noexpr -> Int(1), expenv

    (* Processes a single statement; returns updated environment *)
    in let rec stmt stenv e =
      if stenv.return_val <> None then stenv
      else match e with
          Expr e -> let c, renv = expr stenv e in renv
        | Vdecl(t, id) -> (
            try
              let _ = NameMap.find id stenv.scope.variables
              in raise (Duplicate_name(id ^ " already defined"))
            with Not_found ->
              let typ = type_from_str t in
              let new_entry typ = (
                match typ with
                    SArray -> (Array(arr_type_from_str t, [||]), typ)
                  | _ -> (None, typ)
              )
              in let newval, newtyp = new_entry typ
              in let new_scope = {
                parent = stenv.scope.parent;
                variables =
                  NameMap.add id (newval, newtyp) stenv.scope.variables
              } in {
                glob_env = stenv.glob_env;
                scope = new_scope;
                return_type = stenv.return_type;
                return_val = stenv.return_val
              }
          )
        | VdeclAsn(s, e) -> (*print_endline ("VdeclAsn");*)
            let newstenv = stmt stenv s in
            let c, newenv = expr newstenv e in newenv
        | Return(e) -> (*print_endline "Return";*)
            let c, renv = expr stenv e in
            let stype = c_to_s_type c in
            if  stype <> renv.return_type
            then raise (Type_mismatch(
              "Function type " ^ (str_from_type renv.return_type)
              ^ "does not match function return type " ^ (str_from_type stype)
            ))
            else {
              glob_env = renv.glob_env;
              scope = renv.scope;
              return_type = renv.return_type;
```

```
601                 return_val = c
602               }
603           | Block(sl) -> (*print_endline "Block";*)
604               let block_scope = {
605                 parent = Some(stenv.scope);
606                 variables = NameMap.empty
607               } in
608               let block_env = {
609                 glob_env = stenv.glob_env;
610                 scope = block_scope;
611                 return_type = stenv.return_type;
612                 return_val = stenv.return_val
613               } in
614               let post_block_env = List.fold_left stmt block_env sl in
615               let post_block_scope = ( match post_block_env.scope.parent with
616                   Some(parent) -> parent
617                 | _ -> stenv.scope
618               ) in
619               {
620                 glob_env = post_block_env.glob_env;
621                 scope = post_block_scope;
622                 return_type = stenv.return_type;
623                 return_val = stenv.return_val
624               }
625           | If(e,s_if,s_else) ->
626               let c, ifenv = expr stenv e in (
627                 match c with
628                     Int(x) when x > 0 -> stmt ifenv s_if
629                   | Int(0) -> stmt ifenv s_else
630                   | _ -> raise (
631                       Type_error("Conditional must be of type int instead of "
632                       ^ (c_type_str c))
633                     )
634               )
635           | For(init,cond,iter,st) ->
636               let cinit, initenv = expr stenv init in
637               exec_loop initenv cond iter st
638
639       and exec_loop env cond iter st =
640         let c_cond, condenv = expr env cond in
641         match c_cond with
642           Int(x) when x > 0 ->
643             let loopenv = stmt condenv st in
644             let _, iterenv = expr loopenv iter in
645             exec_loop iterenv cond iter st
646         | Int(0) -> env
647         | _ ->
648             raise (Type_error("For loop conditional must be of type int instead of "
649               ^ (c_type_str c_cond)))
650
651       in stmt env block
652   )
653   (* Function execution *)
654   and exec_fun env fxn args =
655     let params = List.fold_left
656       (
657         fun amap (t, id) -> NameMap.add id (None, type_from_str t) amap
658       ) NameMap.empty fxn.formals
```

```
  in
  let funscope = {
    parent = None;
    variables = params
  } in
  let empty_fun_env = {
    glob_env = env.glob_env;
    scope = funscope;
    return_type = type_from_str fxn.ftype;
    return_val = None
  } in
  try
    let _, fun_env = List.fold_left2
      (
        fun (_, fenv) (t, id) c_expr ->
          try
            update_symtab fenv id c_expr (c_to_s_type c_expr)
          with Type_mismatch(s) ->
            raise (Type_mismatch("Invalid argument to function " ^ fxn.fname ^ ": " ^
    id))
      ) (None,empty_fun_env) fxn.formals args
    in
    let post_fun_env = List.fold_left
      (fun nextenv nextstmt ->
        translate nextstmt nextenv
      ) fun_env fxn.body
    in
    post_fun_env (* DO NOT MIX THIS UP WITH THE CALLER'S ENVIRONMENT *)
  with Invalid_argument(s) -> raise (Invalid_argument(fxn.fname ^ ": Wrong number of
    arguments"))
```

execute.ml

```
(*
Andrew OReilly, Stephanie Huang
*)

open Ast
open Sast
open Compile
open Semantics
open Printf

let execute_prog (vdecls, fdecls) =
  let print_env key (c, s) =
    print_endline (key ^ " = " ^ (Sast.c_type_str c)) in
  let var_map = List.fold_left
    (
      fun vmap (t, id) ->
        if NameMap.mem id vmap
        then raise (Duplicate_name(id ^ " already defined"))
        else NameMap.add id (None, (type_from_str t)) vmap
    )
    NameMap.empty vdecls
  in
  let fun_map = List.fold_left
    (
      fun map fdec -> NameMap.add fdec.fname fdec map
```

```
    )
    NameMap.empty fdecls
  in
  let globals = {
    variables = var_map;
    functions = fun_map;
  } in
  let scp = {
    parent = None;
    variables = NameMap.empty
  } in
  let env = {
    glob_env = globals;
    scope = scp;
    return_type = SInt;
    return_val = None
  } in
  let _, postexec_env =
    if NameMap.mem "compose" env.glob_env.functions
    then (), (exec_fun
      env
      (NameMap.find "compose" env.glob_env.functions)
      []
    )
    else (print_endline "No compose function"), env
  in let ofile = open_out "play.out" in
  NameMap.iter print_env postexec_env.scope.variables;
  fprintf ofile "#!/bin/bash\njava CSVPlayer out.csv";
  close_out ofile
```

parser.mly

```
%{
(*
Parser for CMajor
By PLT Sandwich
Andrew OReilly, Stephanie Huang
*)
open Ast
%}

%token PLUS MINUS TIMES DIVIDE LAYER REPEAT ARRCAT ASSIGN EQ NEQ GT GTE LT LTE EOF
%token LPAREN RPAREN LCBRACE RCBRACE LSBRACE RSBRACE COMMA SEMI RETURN IF ELSE FOR
%token PITCH_LIT
%token <int> INT_LIT
%token <int> PITCH_LETTER
%token <int> PITCH_SIGN
%token <string> ID
%token <string> TYPENAME

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left ARRCAT
%left REPEAT
%left EQ NEQ
%left LT GT LTE GTE
%left PLUS MINUS
```

```
%left TIMES DIVIDE
%left LAYER

%start program
%type < Ast.program> program
%type < Ast.expr> expr
%type < Ast.vdecl> vdecl
%%

program:
  decls EOF { $1 }

decls:
    { [], [] }
  | decls vdecl SEMI { ($2 :: fst $1), snd $1 }
  | decls fdecl { fst $1, ($2 :: snd $1) }

vdecl:
    TYPENAME ID { $1, $2                        (* Variable declaration *) }
  | TYPENAME LSBRACE RSBRACE ID { $1^"[]", $4 (* Array declaration *) }

fdecl:
  TYPENAME ID LPAREN formals_opt RPAREN LCBRACE stmt_list RCBRACE
  {
    {
      ftype = $1;
      fname = $2;
      formals = $4;
      locals = [];
      body = List.rev $7;
    }
  }

formals_opt:
    { [] }
  | formal_list { List.rev $1 }

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

stmt_list:
    { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
    expr SEMI                               { Expr($1) }
  | vdecl SEMI                              { Vdecl(fst $1, snd $1) }
  | vdecl ASSIGN expr SEMI                  { VdeclAsn(Vdecl(fst $1, snd $1), Asn(snd
    $1, $3)) }
  | RETURN expr SEMI                        { Return($2) }
  | LCBRACE stmt_list RCBRACE               { Block(List.rev $2) }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { If($3, $5, Block([])) }
  | IF LPAREN expr RPAREN stmt ELSE stmt    { If($3, $5, $7) }
  | FOR LPAREN expr_opt SEMI expr_opt SEMI expr_opt RPAREN stmt
                                            { For($3, $5, $7, $9) }

expr_opt:
```

```
          { Noexpr }
  | expr { $1 }

expr:
  ID ASSIGN expr                  { Asn($1, $3) }
| LSBRACE actuals_opt RSBRACE     { Arr($2) }
| ID LPAREN actuals_opt RPAREN    { Call($1, $3) }
| expr PLUS   expr                { Binop($1, Add, $3)     (* Arithmetic operators *) }
| expr MINUS  expr                { Binop($1, Sub, $3) }
| expr TIMES  expr                { Binop($1, Mul, $3) }
| expr DIVIDE expr                { Binop($1, Div, $3) }
| expr LAYER  expr                { Binop($1, Layer, $3)   (* Layer operator *) }
| expr REPEAT expr                { Binop($1, Rep, $3)     (* Array init operator *) }
| expr ARRCAT expr                { Binop($1, Arrcat, $3) (* Array concat operator *) }
| expr EQ     expr                { Binop($1, Eq, $3)      (* Equality operators *) }
| expr NEQ    expr                { Binop($1, Neq, $3) }
| expr GT     expr                { Binop($1, Gt, $3) }
| expr GTE    expr                { Binop($1, Gte, $3) }
| expr LT     expr                { Binop($1, Lt, $3) }
| expr LTE    expr                { Binop($1, Lte, $3) }
| ID LSBRACE expr RSBRACE ASSIGN expr
                                  { Arrmod($1, $3, $6)     (* modify array *) }
| ID LSBRACE expr RSBRACE         { Arrget($1, $3)         (* array reference *) }
| literal                         { Lit($1)                (* Lit($1) Any literal.
    Should make
                                                              type-checking easier *) }
| ID                              { Id($1)                 (* Identifier *) }


actuals_opt:
                { [] }
| actuals_list   { List.rev $1 }

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


literal:
  INT_LIT                          { Intlit($1) }
| PITCH_LETTER INT_LIT             { Pitchlit($1, $2) }
| PITCH_LETTER                     { Pitchlit($1, 4) }
| LPAREN expr COMMA expr RPAREN    { Tuple($2, $4) }
```

sast.ml

```
(*
Semantic AST
by Andrew OReilly, Stephanie Huang
*)

open Ast

exception Type_mismatch of string
exception Missing_variable of string
exception Missing_function of string
exception Invalid_type of string

```

```
module StringMap = Map.Make(String) ;;

(* Types for semantic checking *)
type s_type = SInt | SPitch | SDur | SNote | SChord | SPhrase
  | SScore | SArray | None

(* cmajor types *)
type c_type =
    Int of int
  | Pitch of int * int
  | Dur of int * int
  | Note of c_type * c_type
  | Chord of c_type array * c_type
  | Phrase of c_type array
  | Score of c_type array
  | Array of s_type * c_type array
  | None

(* Get the semantic type of a storage type *)
let rec c_to_s_type = function
    Int x -> SInt
  | Pitch (l,o) -> SPitch
  | Dur (a,b) -> SDur
  | Note (p,d) -> SNote
  | Chord (p,d) -> SChord
  | Phrase x -> SPhrase
  | Score x -> SScore
  | Array (s,l) -> SArray
  | None -> None

(* Get string representation of a c_type *)
let rec c_type_str = function
    Int(x) -> "Int(" ^ string_of_int x ^ ")"
  | Pitch(l, o) -> "Pitch("^(string_of_int l)^","^(string_of_int o)^")"
  | Dur(a,b) -> "Dur("^(string_of_int a)^","^(string_of_int b)^")"
  | Note(p,d) -> "Note("^(c_type_str p)^","^(c_type_str d)^")"
  | Chord(p,d) -> "Chord("^(c_type_str (Array(SPitch, p)))^","^(c_type_str d)^")"
  | Phrase(p) -> "Phrase("^(c_type_str (Array(SChord, p)))^")"
  | Score(s) -> "Score("^(c_type_str (Array(SPhrase, s)))^")"
  | Array(s,a) -> "Array("^Array.fold_left (
      fun prev x ->
        if (prev = "") then (c_type_str x) else prev^","^(c_type_str x)
    ) "" a^")"
  | None -> "None"
```

scanner.mll

```
{
(*
Scanner for CMajor
By PLT Sandwich
Andrew OReilly, Stephanie Huang
*)
open Parser
}

(* let pitch = '$' [ 'A'-'G' ] [ '#' 'b' ]? <- Not used, but handy *)
let id = [ '_' 'A'-'Z' 'a'-'z' ] [ '_' 'A'-'Z' 'a'-'z' '0'-'9' ]*
```

```
12 let typename = "int" | "pitch" | "dur" | "note" | "chord" | "phrase" | "score"
13
14 rule token = parse
15   [' ' '\t' '\r' '\n'] { token lexbuf }     (* Whitespace *)
16 | "/*"                 { bcomment lexbuf } (* Block comments (/* */) *)
17 | "//"                 { icomment lexbuf } (* Inline comments (//) *)
18
19 | '('                    { LPAREN }        (* Punctuation *)
20 | ')'                    { RPAREN }
21 | '{'                    { LCBRACE }
22 | '}'                    { RCBRACE }
23 | '['                    { LSBRACE }
24 | ']'                    { RSBRACE }
25
26 | ';'                    { SEMI }
27 | ','                    { COMMA }
28
29 | '='                    { ASSIGN }
30 | '+'                    { PLUS }
31 | '-'                    { MINUS }
32 | '*'                    { TIMES }
33 | '/'                    { DIVIDE }
34
35 | "=="                   { EQ }
36 | "!="                   { NEQ }
37 | '>'                    { GT }
38 | ">="                   { GTE }
39 | '<'                    { LT }
40 | "<="                   { LTE }
41
42 | '^'                    { LAYER }
43 | "**"                   { REPEAT }
44 | "++"                   { ARRCAT }
45
46 (* keywords *)
47 | "if"                   { IF }
48 | "else"                 { ELSE }
49 | "return"               { RETURN }
50 | "for"                  { FOR }
51
52
53 (* Pitch literals are split into separate symbols to make parsing easier *)
54 | '$'                    { ptoken lexbuf }
55 | ['0'-'9']+ as lit      { INT_LIT(int_of_string lit) } (* integers *)
56 | typename as type_str   { TYPENAME(type_str) }         (* type names *)
57 | id as id_str           { ID(id_str) }                 (* identifiers *)
58 | eof                    { EOF }                        (* end-of-file *)
59
60 and bcomment = parse
61   "*/" { token lexbuf }
62   | _    { bcomment lexbuf }
63
64 and icomment = parse
65   '\n'   { token lexbuf } (* Should this be quoted? *)
66   | _    { icomment lexbuf }
67
68 and ptoken = parse
69   | "A"                    { PITCH_LETTER(0) }
```

```
  | "A#" | "Bb"            { PITCH_LETTER(1) }
  | "B"  | "Cb"            { PITCH_LETTER(2) }
  | "B#" | "C"             { PITCH_LETTER(3) }
  | "C#" | "Db"            { PITCH_LETTER(4) }
  | "D"                    { PITCH_LETTER(5) }
  | "D#" | "Eb"            { PITCH_LETTER(6) }
  | "E"  | "Fb"            { PITCH_LETTER(7) }
  | "E#" | "F"             { PITCH_LETTER(8) }
  | "F#" | "Gb"            { PITCH_LETTER(9) }
  | "G"                    { PITCH_LETTER(10) }
  | "G#" | "Ab"            { PITCH_LETTER(11) }
  | 'R'                    { PITCH_LETTER(-1) }
  | _ { token lexbuf }
```

semantics.ml

```
(*
Functions for semantic analysis
Andrew OReilly, Stephanie Huang
*)

open Ast
open Sast

(* Returns true if pitch is valid *)
let is_valid_pitch (letter, oct) = match letter, oct with
    -1, _ -> true
  | l, o when 0 <= l && l <= 11 && 0 <= o && o <= 8 -> true
  | _, _ -> false

let raise_pitch n p = match n, p with
    Int(x), Pitch(l,o) -> (
      let o = o + (l + x) / 12 in
      let l = (l + x) mod 12 in
      Pitch(l,o)
    )
  | _, _ -> raise (Invalid_type("raise_pitch: invalid type"))

(* Convert a pitch literal to an integer value *)
let pitch_to_int p =
  match p with Pitch(l, o) -> 9 + o * 12 + l
    | _ -> raise (Invalid_type("pitch_to_int: invalid type"))


(* Matches a string with correspondin s_type *)
let type_from_str t = match t with
    "int" -> Sast.SInt
  | "pitch" -> Sast.SPitch
  | "dur" -> Sast.SDur
  | "note" -> Sast.SNote
  | "chord" -> Sast.SChord
  | "phrase" -> Sast.SPhrase
  | "score" -> Sast.SScore
  | "int[]" | "pitch[]" | "dur[]" | "note[]" | "chord[]" | "phrase[]"
    | "score[]" -> Sast.SArray
  | _ -> raise (Invalid_type ("\""^t^"\""^" is an invalid type"))

(* get components of Array type *)
```

```
43 let get_arr_type a = match a with Array(typ, dat) -> typ
44   | _ -> raise (Invalid_type ("Get get array type from non-array"))
45
46 let get_arr_data a = match a with Array(typ, dat) -> dat
47   | _ -> raise (Invalid_type ("Get get array data from non-array"))
48
49 (* Check that element and array type match up *)
50 let is_valid_elem elem arr_type =
51   if ((c_to_s_type elem) = arr_type) then true else false
52
53 (* Gets type of array from string *)
54 let arr_type_from_str t =
55   let typ =
56     if String.contains t '[' then
57       if String.contains t ']' then String.sub t 0 (String.length t - 2)
58       else raise (Invalid_type ("\""^t^"\""^" is an invalid array type"))
59     else raise (Invalid_type ("\""^t^"\""^" is an invalid array type"))
60   in type_from_str typ
61
62 let str_from_type = function
63     Sast.SInt -> "int"
64   | Sast.SPitch -> "pitch"
65   | Sast.SDur -> "dur"
66   | Sast.SNote -> "note"
67   | Sast.SChord -> "chord"
68   | Sast.SPhrase -> "phrase"
69   | Sast.SScore -> "score"
70   | Sast.SArray -> "array"
71   | None -> "none"
72
73 (* For reducing fractions *)
74 let rec gcd a_arg b_arg =
75   let a = if a_arg < 0 then a_arg * -1 else a_arg in
76   let b = if b_arg < 0 then b_arg * -1 else b_arg in
77   if a = 0 then b
78   else if b = 0 then a
79   else if a = b then
80     a
81   else if a > b then
82     gcd (a - b) b
83   else
84     gcd a (b - a)
85
86 (* Handles cases of dur / int, int / dur, dur / dur *)
87 let dur_divide (a1, a2) =
88   let n, d = match a1, a2 with
89       Dur(dn,dd), Int(i)          -> dn, dd*i
90     | Int(i), Dur(dn,dd)          -> dd*i, dn
91     | Dur(dn1,dd1), Dur(dn2,dd2) -> dn1*dd2, dd1*dn2
92     | _,_ -> 0,0
93   in
94   let g = gcd n d in
95   Dur(n / g, d / g)
96
97 (* Duration subtraction - returns negative values *)
98 let dur_sub = function
99     Dur(n1,d1), Dur(n2,d2) ->
100      let n1sub = d2 * n1 in
```

```
      let d1sub = d2 * d1 in
      let n2sub = d1 * n2 in
      let n = n1sub - n2sub in
      let g = gcd n d1sub in
      Dur(n / g, d1sub / g)
  | _,_ -> raise (Invalid_type("Error: invalid type. Dur expected."))

(* Same as dur_sub, but returns absolute value *)
let dur_sub_abs = function
    Dur(_,_) as d1, (Dur(_,_) as d2) ->
      ( match dur_sub (d1,d2) with
          Dur(n,d) -> Dur(Pervasives.abs n, d)
        | _ -> raise (Invalid_type("This should never happen"))
      )
  | _,_ -> raise (Invalid_type("Error: invalid type. Dur expected."))

(* Add durations *)
let dur_add d1 d2 = match d1, d2 with
    Dur(n1,d1), Dur(n2,d2) -> (
      let n = n1 * d2 + n2 * d1 in
      let d = d1 * d2 in
      let g = gcd n d in
      Dur(n / g, d / g)
    )
  | _, _ -> raise (Invalid_type("Error: invalid type. Dur expected."))
```

Makefile

```
# Andrew O'Reilly, Jonathan Sun

OBJS = ast.cmo parser.cmo scanner.cmo sast.cmo semantics.cmo compile.cmo execute.cmo
    cmajor.cmo

# Choose one
YACC = ocamlyacc
# YACC = menhir --explain

SRCS = $(wildcard *.*)
JAVA_SRCS = $(shell find java -name '*')
TESTS := $(shell find tests -name '*.cmajor' -o -name "*.out")
DEMOS = $(shell find finaldemo *.cmajor -name '*')

TARFILES = Makefile $(SRCS) $(JAVA_SRCS) $(TESTS) $(DEMOS)



cmajor : $(OBJS) CSVPlayer
  ocamlc -g -o cmajor $(OBJS)

CSVPlayer :
  javac CSVPlayer.java

scanner.ml : scanner.mll
  ocamllex scanner.mll

parser.ml parser.mli : parser.mly
  $(YACC) parser.mly

```

```
%.cmi : %.mli
  ocamlc -g -c $<

%.cmo : %.ml
  ocamlc -g -c $<

.PHONY : clean
clean :
      rm -f cmajor parser.ml parser.mli parser.ouput scanner.ml \
      *.cmo *.cmi *.out *.diff *.log *.txt *.gz

.PHONY : cleanJava
cleanJava :
  rm -f java/function_* java/output.java java/*.class


.PHONY : test
test : cmajor testall.sh
  ./testall.sh


.PHONY : all
all : clean cmajor CSVPlayer



# Generated by ocamldep *.ml *.mli >> Makefile
ast.cmo :
ast.cmx :
cmajor.cmo : parser.cmi execute.cmo compile.cmo
cmajor.cmx : parser.cmx execute.cmx compile.cmx
compile.cmo : semantics.cmo sast.cmo ast.cmo
compile.cmx : semantics.cmx sast.cmx ast.cmx
execute.cmo : sast.cmo compile.cmo
execute.cmx : sast.cmx compile.cmx
parser.cmo : ast.cmo parser.cmi
parser.cmx : ast.cmx parser.cmi
sast.cmo : ast.cmo
sast.cmx : ast.cmx
semantics.cmo : sast.cmo ast.cmo
semantics.cmx : sast.cmx ast.cmx
parser.cmi : ast.cmo # this was extra, after an unsuccessful make.
```

CSVPlayer.java

```
/**
 *  Jonathan Sun
 */


import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Scanner;

public class CSVPlayer {

    final static int MEASURE_DUR = 2000; // Default measure duration, milliseconds.
```

```
14      final static int PITCH = 0;
15      final static int DUR = 1;
16
17      public static void printUsage() {
18          System.err.println(
19              "Usage: java CSVPlayer </my/path/to/file.csv>"
20          );
21          System.exit(1);
22      }
23
24      public static int[] calcDurs(int[] dur1s, int[] dur2s) {
25          int[] durs = new int[dur1s.length];
26          for (int i = 0; i < dur1s.length; i++)
27              durs[i] = (int)((double)MEASURE_DUR * ((double)dur1s[i] / (double)dur2s[i])
    );
28          return durs;
29      }
30
31      /**
32       * Converts a string of comma separated integers to an array of integers.
33       * @param csv comma separated integers
34       * @return integer array
35       */
36      public static int[] scanInts(String csv) {
37          String[] nums = csv.split(",");
38          int[] ints = new int[nums.length];
39          for (int i = 0; i < nums.length; i++)
40              ints[i] = Integer.parseInt(nums[i]);
41          return ints;
42      }
43
44      public static ArrayList<int[][]> buildScore(Scanner input) {
45          ArrayList<int[][]> score = new ArrayList<int[][]>();
46
47          /* no error handling... */
48          while (input.hasNextLine()) {
49              int[][] voice = new int[2][];
50              int[] pitches = scanInts(input.nextLine());
51              int[] dur1s = scanInts(input.nextLine());
52              int[] dur2s = scanInts(input.nextLine());
53              int[] durs = calcDurs(dur1s, dur2s);
54
55              voice[PITCH] = pitches;
56              voice[DUR] = durs;
57
58              score.add(voice);
59          }
60          return score;
61      }
62
63      public static void play(ArrayList<int[][]> score) {
64          ArrayList<NotesPlayer> voices = new ArrayList<NotesPlayer>();
65          for (int[][] voice : score)
66              voices.add(new NotesPlayer(voice));
67          for (NotesPlayer notes : voices)
68              (new Thread(notes)).start();
69      }
70
```

```
    public static void main(String[] args) throws FileNotFoundException {
        if (args.length != 1) printUsage();

        Scanner input = new Scanner(new File(args[0]));
        ArrayList<int[][]> score = buildScore(input);
        input.close();

        play(score);

        System.out.println(
            "Thank you for your patronage. Have a nice day."
        );
    }
}
```

NotesPlayer.java

```
/**
 * Jonathan Sun
 */

import javax.sound.midi.MidiSystem;
import javax.sound.midi.MidiChannel;
import javax.sound.midi.MidiUnavailableException;
import javax.sound.midi.Synthesizer;
import java.lang.Runnable;
import java.lang.Thread;

public class NotesPlayer implements Runnable {

    final static int VOLUME = 100;
    final static int PITCH = 0;
    final static int DUR = 1;
    final static int MEASURE_DUR = 2000; // Default measure duration

    public NotesPlayer(int[][] notes) {
        this.notes = notes;
    }

    @Override
    public void run() {
        try {
            playNotes();
        } catch (MidiUnavailableException e) {
            threadMessage("Midi Unavailable");
        } catch (InterruptedException e) {
            threadMessage("Thread interrupted.");
        }
    }

    private void playNotes() throws MidiUnavailableException, InterruptedException {
        Synthesizer synth = MidiSystem.getSynthesizer();
        MidiChannel[] channels = synth.getChannels();
        synth.open();
        for (int i = 0; i < notes[0].length; i++) {
            System.out.println("Playing: " + notes[PITCH][i]);
            if (notes[PITCH][i] == -1) {
                Thread.sleep(notes[DUR][i]);
```

```
            } else {
                channels[0].noteOn(notes[PITCH][i], VOLUME);
                Thread.sleep(notes[DUR][i]);
                channels[0].noteOff(notes[PITCH][i], VOLUME);
            }
        }
        synth.close();
    }

    private void threadMessage(String message) {
        String threadName = Thread.currentThread().getName();
        System.out.format("%s: %s%n", threadName, message);
    }

    private int[][] notes;
}
```

8.2 Demos

Row Your Boat

```
int compose() {
  pitch[] pitches = $C ** 3
    ++ $D ++ $E
    ++ $E ++ $D ++ $E ++ $F ++ $G
    ++ $C ** 3 ++ $G ** 3
    ++ $E ** 3
    ++ $C ** 3
    ++ $G ++ $F ++ $E ++ $D ++ $C;

  dur dot8 = (3,16);
  dur trip8 = (1,4) / 3;

  dur[] durations = (1,4) ** 2
    ++ dot8 ++ (1,16) ++ (1,4)
    ++ dot8 ++ (1,16) ++ dot8 ++ (1,16)
    ++ (1,2)
    ++ trip8 ** 12
    ++ dot8 ++ (1,16) ++ dot8 ++ (1,16)
    ++ (1,2);

  phrase mainphrase = pitches ^ durations;

  note rest = ($R,(1,1));

  int i;
  score song = newscore();

  for(i = 0; i < 4; i = i + 1) {
    int j;
    phrase round = mainphrase;
    for(j = 0; j < i; j = j + 1) {
      round = rest + round;
    }

```

```
    song = song ^ round;

  }

  play(song);

}

chord newchord(dur d) {
  note n1 = ($R,d);
  return n1 ^ $R;
}

phrase newphrase() {
  chord c1 = newchord((0,1));
  chord c2 = c1;
  return c1 + c2;
}

score newscore() {
  phrase p1 = newphrase();
  return p1 ^ p1;
}
```

Shepard Scale

```
int compose() {
  dur d = (1,8);
  pitch[] pitches = $C ** 8;
  pitch base = $C0;
  pitch max = $C8;
  int i;
  int n = 40;

  //Initialize an array of pitches
  for(i = 0; i < 8; i = i + 1) {
    pitches[i] = base + i * 12;
  }

  //Main loop
  phrase ph = newphrase();
  for(i = 0; i < n; i = i + 1) {
    int j;
    for(j = 0; j < 8; j = j + 1) {
      pitches[j] = pitches[j] + 1;
      if(pitches[j] == max)
        pitches[j] = base;
    }

    //Put them all on top of one another
    chord ch = newchord(d);
    for(j = 0; j < 8; j = j + 1) {
      ch = ch ^ pitches[j];
    }

    ph = ph + ch;
  }

```

```
33    play(ph);
34
35  }
36
37  chord newchord(dur d) {
38    note n1 = ($R,d);
39    return n1 ^ $R;
40  }
41
42  phrase newphrase() {
43    chord c1 = newchord((0,1));
44    chord c2 = c1;
45    return c1 + c2;
46  }
```