m

A language for music generation.

Final Report

Yiling Hu (yh2378)
Monica Ramirez-Santana (mir2115)
Jiaying Xu (jx2129)

Table of Contents

1. Introduction

m is a language specifically designed for algorithmic music composition. The language allows a programmer or composer to write algorithms that generate music.

Traditionally, composers would have to sequentially write out every note of their composition, one by one. However, using this language the composer could specify sets of notes and chords that could be used in conjunction with arithmetic operators, control structures, and randomization functions to algorithmically generate music. The arithmetic manipulation of musical concepts is achieved through the translation of all traditional musical concepts into arithmetic types, `int` and `float`.

Many interesting uses for the language exist. It could be interesting to study purely mathematical concepts through music. For example, what does the Fibonacci sequence sound like as music? Another exciting real world application of the language could be to generate music for Pandora's (www.pandora.com) user radio stations. The generated music would add novelty and diversity. Stations would no longer be limited to the realm of recorded songs. Instead, each song would be different from any that the user had heard before. This would solve the problem of users becoming bored with their radio stations.

2. Language Tutorial

This section contains a brief introduction to the m language. It is intended as a tutorial on the language, and aims at getting a reader new to m started as quickly as possible.

2.1 Hello World

Let's start by composing a song with a single note – middle C – played on the piano. Type the following program into your favorite editor:

```
/*      The 'Hello World' program of the m algorithmic music composition
        language. Plays middle C. */

void main()
{
        note n;
        chord c;
        staff s;
        part p;

        n.pitch = C4;                   // set the note, middle C
        n.duration = 1.0;
        n.intensity = 100;

        s.instrument = 0;
```

```
        p.bpm = 60;
        p.beatsig = 0.25;

        add(c, n);
        add(s, c);
        add(p, s);

        play(p);
}
```

2.1.1 Compiling and Running

Save the code in the file hello.m, then compile it by typing:

```
m < hello.m
```

This creates an executable file output.java. Now compile this by typing:

```
javac Output.java
```

This creates an executable output.class which is then executed by typing:

```
java Output
```

The result is that a MIDI file output.mid is created. You can execute this file with your favorite music player. The output of this executable is the sound of middle C being played on a piano.

2.1.2 Anatomy of an m Program

An m program contains functions and variables. The functions specify the tasks to be performed by the program. The "main" function establishes the overall logic of the code. It calls different functions to perform the necessary sub-tasks. All m programs must have a "main" function.

The void preceding "main" indicates that main is of `void` type--that is, it has *no* type associated with it, meaning that it cannot return a result on execution.

Our hello.m code calls add, a standard library function which adds the second argument to the collection field of the first argument. It also calls play, a standard library function which outputs the argument to the output.mid file. The standard library contains these two functions as well as the randint, randfloat, and print functions; the library is automatically included in every m program. This means you can always call or use these functions in any m program. See Appendix B for the full standard library.

The variables in our hello.m are n, c, s, and p of types `note`, `chord`, `staff`, and `part`. These types are based on traditional musical concepts.

Notes consist of a pitch, intensity (volume), and duration. When a note is played, its pitch is played at the intensity defined for the duration defined. The pitch of our note n is C4, which is shorthand for describing C in the fourth octave. To achieve the same result, the value of 60 could have also been assigned to n's pitch. See Appendix C for a complete listing of note values.

Notes are added to a chord by using the add function. A chord is defined as a list of notes. When a chord is played, all of its notes are played at the same time.

Chords are added to staffs by using the add function. Staffs consist of a list of chords and an instrument. When a staff is played, the chords are played on the instrument defined in the order that they were added. Our staff has instrument set to 0, which is a piano. See Appendix C for a complete listing of instrument values.

Staffs are added to parts by using the add function. Parts consist of a list of staffs, a bpm (beats per minute), and a beat signature. When play is called on a part, all of its staffs are played simultaneously. This feature of the language can be used to mimic the right hand and left hand sections of a piano piece or the different instruments of a multi-instrument ensemble. The bpm field helps calculate the duration each note should be played. At a bpm of 60, one beat should be played every second. With a beat signature of 0.25, we should play a quarter note every beat. This means that our middle C with duration 1.0 will be played for 4 seconds.

You can only call play on a part, so even in our simple program that plays one note, we must define the note, add it to a chord, add that to a staff, and add that to a part. If play is called more than once in a program, the Output.java is rewritten every time. This means that only the last invocation of play has any effect on the output.mid file.

Type	**Members**	**add()**	**play()**
`note`	pitch intensity duration	Cannot add anything to type `note`	Play the pitch at the intensity defined for the duration defined.
`chord`	Collection of notes	Add type `note`	Play its notes simultaneously.
`staff`	Collection of notes and chords instrument	Add types `note` and `chord`	Play its chords in the order they were added.
`part`	bpm beat signature	Add type `staff`	Play its staffs simultaneously.

Note: when a chord C is played, the next chord in the staff will begin playing after the first note in the chord C's list has finished playing.

The ";" denotes the end of a statement. Blocks of statements are put in braces {...}, as in the definition of functions. All m statements are defined in free format, i.e., with no specified layout or column assignment. Whitespace (newlines, tabs, or spaces) is never significant, except when ending a single line // comment which needs a newline to terminate, and when distinguishing between otherwise indistinguishable tokens. The following program would produce exactly the same result as our earlier example:

```
/*The 'Hello World' program of the m algorithmic music composition language.
Plays middle C. */void main(){note n;chord c;staff s;  part p;n.pitch = C4;// set
the note, middle C
n.duration = 1.0;n.intensity = 100;s.instrument = 0;p.bpm = 60;p.beatsig =
0.25;add(c, n);add(s, c);add(p, s);play(p);}
```

The reasons for arranging your programs in lines and indenting to show structure should be obvious!

2.2 Computation

test.m:

```
/*      Testing the arithmetic functionality of m. */

void main()
{
        //Testing integer arithmetic
        print( 1 + 2 );
        print( 1 - 2 );
        print( 2 * 2 );
        print( 1 / 2 );
        print( 5 % 2 );

        //Testing floating point arithmetic
        print( 1.0 + 2.0 );
        print( 1.0 - 2.0 );
        print( 2.0 * 2.0 );
        print( 1.0 / 2.0 );

        //Order of operations with integer arithmetic
        print( 1 + 2 * 3 + 4 );
        print( ( 1 + 2 ) * ( 3 + 4 ) );
}
```

The above program displays the arithmetic capability of the m language. We use the print standard library function to display output. The output of the print function will be displayed upon compiling the test.m with the command 'm < test.m'. Note that because we do not call play on a part, there will be no Output.java file, so this program will not ultimately create a MIDI file that plays a song.

The +, -, *, / operators are available for the programmer to use with either integer or floating point numbers; % can only be used with integers. Note that in the output, 1/2 evaluates to 0. This is because the return value of an integer arithmetic expression is an integer. Therefore, the 0.5 result is truncated to 0. The order of operations is the normal order of operations, and a different precedence may be forced as usual with the use of ().

2.3 Selection

```
/*      Testing the if statement in m. */

void main()
{
        if(true)
        {
                print( 0 );
        }
```

```
        if(false)
        {
                print( 1 );
        }
        else
        {
                print( 2 );
        }
}
```

The above program displays the selection capability of the m language.

if statement:

if(cond-expression) { statement-block }

if(cond-expression) { statement-block } else { statement-block }

The if statement executes the statement-block only if the cond-expression evaluates to true. If the optional else statement is included, execution will fall through to the second-statement block if the cond-expression evaluates to false.

Note that m supports the `bool` type as shown by the keywords “true” and “false” in the program. The !, &&, and || operators are supported by m as well as the typical relational operators (i.e. ==, !=, <, <=, …). These all have the typical precedence and associativity and evaluate to `bool` values.

2.4 Iteration

```
/*      Testing the for statement in m. */

void main()
{
        int i;

        for( i = 0; i < 10; i++)
        {
                print(i);
        }
}
```

```
/*      Testing the while statement in m. */

void main()
{
        int i;
        i = 0;

        while( i < 10 )
        {
                print(i);
                i++;
        }
}
```

The above programs display the iteration capabilities of the m language.

for statement:

for(init-statement; cond-expression; loop-expression) { statement-block }

The for loop executes statement-block and loop-expression repeatedly until cond-expression becomes false. Use the for statement to construct loops that must execute a specified number of times.

while statement:

while(cond-expression) { statement-block }

The while loop executes statement-block repeatedly until cond-expression becomes false. Note that because cond-expression in while is evaluated before statement-block is executed, statement-block may be executed 0 or more times.

2.5 Functions

Recall that in our hello.m program we called the standard library functions add and play. You can also create your own user-defined functions in an m program. Note that because the standard library functions are predefined and included in every m program, you may not name any of your user-defined functions any of the following: add, play, randint, randfloat, print, main.

```
/*        Testing the function capability in m. */

int const( )
{
        return 0;
}

void prnt( int a, int b )
{
        print(a);
        print(b);
}

int plus( int a, int b )
{
        return a + b;
}

void main()
{
        print(const());
        prnt( 1, 2 );
        print(plus( 1, 2 ));
}
```

The first function we have defined is named const. It returns type `int` and takes two arguments of type `int`, a and b. Within the code block which defines the execution of this function, we return 0.

The second function we have defined is named prnt. It returns type `void` and takes two arguments of type `int`, a and b. Within the code block which defines the execution of this function, we call the standard library function print on a and b. Note that we do not return anything. This is because you

cannot return from a void function. Including a return statement in a void function (including main) will result in an error.

The last function we have defined is named plus. It returns type `int` and takes two arguments of type `int`, a and b. Within the code block which defines the execution of this function, we return the sum of a and b. Note that you refer to the arguments passed to the function by the variable names used in the definition of the function.

3. Language Reference Manual

The following describes the syntax for the m language.

3.1 Lexical Conventions

An m program consists of the elements discussed below. You use these elements, called “lexical elements” or “tokens” to construct statements, definitions, declarations, and so on, which are used to construct complete programs.

3.1.1 Tokens

There are five types of tokens: identifiers, keywords, operators, literals, and separators.

Spaces, tabs, and newlines (collectively, “white space”) are ignored except when used as separators. Separators are white space that is needed to separate otherwise adjacent identifiers, keywords, and pitch constants. (Comments, as described below, are also ignored.)

If the input stream has been separated into tokens up to a given character, the next token is the longest string of characters that could constitute a token.

3.1.2 Comments

There are two methods to insert a comment: (1) the characters /* introduce a comment, which terminates with the characters */ these types of comments do not nest. (2) the characters // introduce a comment, which terminates at the following newline character.

3.1.3 Identifiers

An identifier is a sequence of letters (including _) and digits, the first character of which is a letter a-z or A-Z. Identifiers may be of any length. In identifiers, upper- and lowercase letters are different.

3.1.4 Keywords

The following identifiers are reserved for the use as keywords, and may not be used otherwise:

`part`	`int`	`for`
`staff`	`float`	`while`
`note`	`bool`	`return`
`chord`	`if`	`true`
`void`	`else`	`false`

3.1.5 Literals

There are four types of literals: integer literals, float literals, Boolean literals, and pitch literals.

literal:

integer-literal
floating-literal
boolean-literal
pitch-literal

Integer Literals

An integer literal consists of a sequence of digits and is always interpreted as a decimal number.

Floating Literals

A floating literal consists of an integer part, a decimal part, and a fraction part.

$$[integer].[fraction]_{opt}$$

The integer and fraction parts both consist of a sequence of digits. The fraction part is optional but the integer and decimal part must always be included.

Pitch Literals

A pitch literal is a sequence of characters and digits in one of the two formats outlined below:

Either:

$$[note][modifier]_{opt}[octave]$$

note: accepts the values a-g and A-G.

modifier: accepts the values s, f, S, F. May be omitted.

octave: accepts the values 0-9.

Or:

The value r or R.

Examples of pitch constants: As7, af7, A7, r

3.2 Identifiers

Identifiers are names that refer to functions and objects. An object is a named region of storage, and a variable is an identifier that refers to an object.

A variable has a scope and type. A scope determines the lifetime of the storage associated with the variable. The type indicates how the data contained at the storage location is interpreted.

3.2.1 Scope

Scope is the region of the program in which a variable is known.

There are two kinds of scopes: static and regular. The context of an object's declaration determines its scope. Static variables are all variables that are defined outside of any blocks within the program. Blocks are any segment of code encased in {} braces. Regular variables are all variables that are defined inside of a block.

Static variables are globally known. Regular variables are known inside of the block in which they are defined as well as all blocks contained within that block.

All variables are known within their scope only after their declaration.

3.2.2 Types

There are four basic types: void, integer, floating point, and Boolean, and five derived types: note, chord, staff, part, function.

type:

void
int
float
bool
note
chord
staff
part

Basic Types

The void type specifies an empty set of values and is denoted by the keyword `void`.

The integer type specifies a signed integer and is denoted by the keyword `int`. Default value is 0.

The floating point type specifies a signed floating point number and is denoted by the keyword `float`. Default value is 0.0.

The Boolean type specifies a truth value of either true or false and is denoted by the keyword `bool`. Default value is `false`.

Derived Types

The `note` derived type contains three objects: pitch, duration, and intensity. The pitch object accepts pitch constants and integer values 0-128; default value is 128. The duration object accepts floating point values in the range 0-1 inclusive; default value is 0. The intensity object accepts integer values 0-100; default value is 0.

The `chord` derived type contains one object: a collection of `note`s. Default value is an empty set of `note`s.

The `staff` derived type contains three objects: BPM, beat signature, and a collection of `chord`s. Default value of the collection is an empty set of `chord`s. The BPM object accepts integer values 0-240; default value is 0. The beat signature object accepts floating point values in the range 0-1 inclusive; default value is 0.

The `part` derived type contains two objects: instrument and a collection of staffs. Default value of the collection is an empty set of `staff`s. The instrument object accepts integer values 0-127; default value is 0.

The function derived type returns objects of a given type.

3.3 Objects lvalues, and rvalues

An object, also referred to as an rvalue, is a named region of storage; an lvalue is an expression referring to an object. An obvious example of an lvalue expression is an identifier with suitable type and storage class.

The names “lvalue” and “rvalue” come from the assignment expression E1 = E2 in which the left operand E1 must be an lvalue expression and the right operand E2 must be an rvalue expression.

3.4 Expressions

Expressions are identifiers; literals; assignment applied to expressions; member access, prefix, postfix, or binary operators applied to expressions; or expressions in parentheses.

expression:
identifier
literal
expression member-access-op expression
expression assignment-op expression
expression binary-op expression
prefix-op expression
expression postfix-op
(expression)

A parenthesized expression is an expression whose type and value are identical to those of the unadorned expression. The operators that may be applied to expressions are described in the sections below, in rough order of their precedence. Each section describes a particular type of operator and its associativity. See the *m Operator Precedence and Associativity* table at the end of this chapter for a more detailed discussion of operator precedence and associativity.

3.4.1 Member Access Operator

The member access operator groups left to right.

member-access-op:

.

The operands of the member access operator must be identifiers. The result of a member access of the form A.B is the value of the object B contained within the object A. The type of the result is the type of the object B. Member access may not be applied successively (i.e. A.B.C is not allowed).

3.4.2 Postfix Operators

The postfix operators group left to right.

postfix-op: one of

++ -- *(argument-expression-list$_{opt}$)*

argument-expression-list:

expression

argument-expression-list , expression

++ and --

The ++ postfix expression of the form expression++ is equivalent to expression = expression + 1. The -- postfix operator is analogous to ++. The operand to ++ and -- must therefore be an lvalue.

Function Calls

A function call is a postfix expression. A function call is made in the following way: the function designator followed by parentheses containing a possibly empty, comma-separated list of expressions which constitute the arguments to the function. The function call has the type of the function return type. A function must be defined before it is called.

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. The effect of the call is undefined if the number of arguments disagrees with the number of parameters and if the types of the arguments disagree in the definition of the function. Parameters are passed by value into the function. The order of evaluation of arguments is unspecified. However, the arguments and the function designator are completely evaluated, including all side effects, before the function is entered. Recursive calls to any function are permitted.

3.4.3 Prefix Operator

The prefix operator groups right-to-left.

prefix-operator:

!

The operand of the ! operator must have `bool` type, and the result is true if the value of its operand compares equal to false, and false otherwise. The type of the result is `bool`.

3.4.4 Binary Operators

There are several kinds of binary operators; all group left to right.

binary-op:
arithmetic-op
relational-op
logical-op

Arithmetic Operators

arithmetic-op: one of
`*  /  %  +  -`

The operands of the arithmetic operators must be of the same arithmetic type (i.e. both `int` or both `float`, `int` + `float` is not accepted). The operands of % must be of type `int`.

The binary * operator denotes multiplication. The binary / operator yields the quotient, and the % operator the remainder, of the division of the first operand by the second. If the second operand is 0, the result is undefined. The type of the result is the same as the type of the operands.

The operands of + and - must be of the same arithmetic type (i.e. both `int` or both `float`, `int` + `float` is not accepted). The result of the + operator is the sum of the operands. The result of the - operator is the difference of the operands. The type of the result is the same as the type of the operands.

Relational Operators

relational-op: one of
`<  >  <=  >=  ==  !=`

The operands of < (less), > (greater), <= (less or equal) and >= (greater or equal) must be of arithmetic type. Note that unlike with the arithmetic operators the operands aren't required to be of the same arithmetic type (e.g. `int` < `float` is allowed). The result is a true or false `bool` values.

The operands of == (equal to) and the != (not equal to) operators must be of type `bool`. The result is a true or false `bool` value.

Logical Operators

logical-op: one of
&& ||

The operands of && (logical AND) need to be of type `bool`. It is important to note that && guarantees left-to-right evaluation: the first operand is evaluated, including all side effects. It returns true if both its operands compare unequal to false, false otherwise. The result is true or false `bool` value.

The operands of || (logical OR) need to be of type `bool`. It returns true if either of its operands compare unequal to false, and false otherwise. || guarantees left-to-right evaluation: the first operand is evaluated, including all side effects. If it is equal to false, the value of the expression is false. Otherwise, the right operand is evaluated, and if it is unequal to false, the expression's value is true, otherwise false. The result is true or false `bool` values.

3.4.5 Assignment Operators

There are several kinds of assignment operators; all group right to left.

assignment-op: one of
*= *= /= %= += -=*

An assignment operation assigns the value of the right-hand operand to the storage location named by the left-hand operand. Therefore, the left-hand operand of an assignment operation must be a modifiable lvalue. After the assignment, an assignment expression has the value of the left operand but is not an lvalue.

An expression of the form E1 op= E2 is equivalent to E1 = E1 op (E2) except that E1 is evaluated only once.

3.4.6 m Operator Precedence and Associativity

The following table lists m operators in order of precedence (highest to lowest).
Their associativity indicates in what order operators of equal precedence in an expression are applied.

Operator	Description	Associativity
. ++, --	Member access via object name Postfix increment/decrement	left to right left to right
!	Prefix logical negation	right to left
*,/,%	Multiplication/division/modulus	left to right
+,-	Addition/subtraction	left to right
<, <= >, >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left to right left to right
==, !=	Relational equal to/not equal to	left to right
&&	Logical AND	left to right
||	Logical OR	left to right

= +=, -= *=, /=, %=	Assignment Addition/subtraction assignment Multiplication/division/modulus assignment	right to left

Note: Parentheses are also used to group sub-expressions to force a different precedence; such parenthetical expressions can be nested and are evaluated from inner to outer.

3.5 Declarations

Declarations specify the interpretation given to each identifier; they do not reserve storage associated with the identifier. Declarations have the form:

declaration:
type identifier;

Only one object may be declared per declaration-statement. There are no function declarations. Function declarations not allowed, since functions must be defined at the time they are declared.

3.6 Statements

Except as described, statements are executed in sequence. Statements are executed for their effect, and do not have values. They fall into several groups:

statement:
expression;
compound-statement
selection-statement
iteration-statement
return $expression_{opt}$*;*

Most statements are expression statements, which are assignments or function calls. All side effects from the expression are completed before the next statement is executed.

3.6.1Compound Statement

compound-statement:
{ $statement\text{-}list_{opt}$ *}*

statement-list:
statement
statement-list statement

So that several statements can be used where one is expected, the compound statement (also called “block”) is provided. The body of a function definition is a compound statement.

3.6.2 Selection Statements

Selection statements are used for flow control.

selection-statement:
if(expression) compound-statement
if(expression) compound-statement else compound-statement

3.6.3 Iteration Statements

Iteration statements specify looping.

iteration-statement:
while(expression) compound-statement
for(expression$_{opt}$; expression$_{opt}$; expression$_{opt}$) compound-statement

3.7 Function Definitions

Function definitions have the form:

function-definition:
type identifier(parameter-list$_{opt}$) { declaration-list$_{opt}$ statement-list$_{opt}$ }

parameter-list:
type identifier
parameter-list, type identifier

declaration-list:
declaration
declaration-list declaration

A function may return `void` type, an arithmetic type `int` or `float`, `bool` type, or the derived types `note`, `chord`, `staff`, and `part`.

3.8 m Program

An m program consists of a single file with the form:

program:
declaration-list$_{opt}$ function-definition-list$_{opt}$

function-definition-list:
function-definition
function-definition-list function-definition

At least one of the program's functions must be named "main". This function is what will be executed when the m program is run.

4. Project Plan

4.1 Planning, Specification, Development, Testing Process

Planning

When the group was initial formed, we had weekly meetings to discuss the type of language we wanted to create and the features of that language that would make it better over what is currently used. After deciding on a music generation language, we listed out the different features that were essential to making such a language work, and also some of the features which would be nice to have. We knew in order for algorithms composition to work, we needed to support at least the basic mathematical operations. We also decided that the easiest way to compose music would be to base it off of standard music composition rules. Thus, we needed to support integers and floating point numbers. In order to store data, we found that the built-in support for lists in Ocaml was quite extensive and would be able to perform many of the tasks we needed to.

Specification

After deciding what we wanted to do, we set out to define a formal specification for our language. We started with the C Language Reference Manual and based our specification on that. This was always done at group meetings so that any objections or new ideas that came up could immediately be voiced, discussed, and shot-down/supported by a majority vote. At this point, we included all the features that we wanted our language to have, including ones that were not essential to its function but would provide convenience to any users of our language.

Development

After losing a team member due to dropping out of the class, we set on to continue our project. We started the writing our scanner and parser based on our grammar. We used the scanner and parser files from MicroC as a starting point and added our own pieces where we needed them. When it came time to write the interpreter, we though there were some issues with our original parser in terms of being able to support and check for different types. We changed it a few times, before finally realizing that our original version was the closest to what we needed. At this point, we also dropped some of the features that we wanted to have that would have made it more convenient for programmers using our language, but were ultimately too costly to implement. We also researched different libraries to write to MIDI, which was our target executable file, and found that Java would be the easiest to use and compile to.

Testing

When our interpreter was more or less completed, we began testing the arithmetic and algorithmic functionalities of it. Since we did not have a way to convert any of our language into actual music without the compiler, we started by testing if basic math operations, and control flow statements performed the way we wanted them to. After our compiler was finished, and we had a way to write into MIDI, we tested our songs by listening to their MIDI outputs.

4.2 Programming Style Guide

The following is a list of programming styles we used in our project.

- Provide comments and general code description to make the code understandable to all team members.
- Regularly and frequently check in code to the repository
- Always perform an update from the repository before a commit to minimize conflicts
- Use clear and relevant variable and function names throughout the code
- Notify all team members when new code was added to the repository
- Release code in stages so other team members can begin using parts of your code without having to wait for the complete version
- Provide accurate comments when committing/updating files in the repository
- Inform the team when a bug was found in the code.

4.3 Project Timeline

September 29: Project Proposal

- Concept behind the language, motivation
- Rough idea of syntax
- Language white paper written

November 3: Language Reference Manual

- scanner, parser, and ast 1.0 developed
- Syntax rigorously defined
- Language Reference Manual written

December 22: Complete Compiler, Final Report

- Interpreter and translation into Java source code completed
- Testing completed
- LRM modifications completed
- Tutorial created
- All other documentation completed

4.4 Responsibilities

All team members participated in all phases of the project; however, the following is a rough listing of responsibilities that each team member took the lead on:

> Yiling Hu: understanding the structure of a Java program that produces the .mid file, implementing the code writing to the Java source file

Monica Ramirez-Santana: developing grammar; developing scanner, parser, and ast; Makefile; testing; documentation

Jiaying Xu: project manager; developing the interpreter

4.5 Software Development Environment

Languages Used: Objective CAML, Java 1.6

Tools: Eclipse, Notepad++, javax.sound.midi libraries

Version Control System:

Repository was hosted on Google Code with SVN version control system. Each client used Tortoise SVN to update and commit code.

System Requirements:

"m" music language requires the latest versions of Objective CAML, and Java.

4.6 Project Log

Rev	Commit log message	Date	Author
r79	added sample music programs	Today (5 hours ago)	Yiling
r78	[No log message]	Today (8 hours ago)	Monica
r77	[No log message]	Today (8 hours ago)	Monica
r76	[No log message]	Today (9 hours ago)	Yiling
r75	[No log message]	Today (9 hours ago)	Yiling
r74	added support for multiple instruments	Today (11 hours ago)	Yiling
r73	added support for chords vs notes.	Today (12 hours ago)	Yiling
r72	[No log message]	Today (15 hours ago)	Yiling
r71	tests moved	Dec 19 (3 days ago)	Monica
r70	tests deleted	Dec 19 (3 days ago)	Monica
r69	tests modified	Dec 19 (3 days ago)	Monica
r68	deleted ast.mli	Dec 17 (4 days ago)	Yiling
r67	added ast.ml	Dec 17 (4 days ago)	Yiling
r66	deleted ast.ml	Dec 17 (4 days ago)	Yiling
r65	'hello world' program created	Dec 17 (5 days ago)	Monica
r64	[No log message]	Dec 17 (5 days ago)	Monica
r63	Current bugs: 1. randint, randfloat not truly random. Compile the same file more than once, you'll get the same 'random' value every time. 2. Regular "return;" statement is interpreted as returning type Boolean. Should be interpreted as returning	Dec 17 (5 days ago)	Monica

	type void.		
r62	Test files created. Current bugs: 1. randint, randfloat not truly random. Compile the same file more than once, you'll get the same 'random' value every time. 2. Regular "return;" statement is interpreted as returning type Boolean. Should be interpreted as returning type void. 3. Getting the following error: "Fatal error: exception Failure("cannot assign: note = bool")" for the following code snippet: /* Testing the functionality of m built-in objects. */ note create note(int a, float b, int	Dec 17 (5 days ago)	Monica
r61	compile.ml taken out of Makefile altogether; if, if/else precedence bug fixed	Dec 16 (5 days ago)	Monica
r60	Makefile modifications for Windows, use as is; for Unix uncomment 9,10,32-35 and comment 6,7,27-30	Dec 16 (5 days ago)	Monica
r59	Makefile syntax corrected. For Windows, use as is. For Unix, comment out lines 24-27 and uncomment lines 29-32.	Dec 16 (5 days ago)	Monica
r58	Created a Makefile that compiles scanner, ast, parser, interpreter. Errors at ocamlc -c compile.ml, probably because compile.ml is old and buggy. For Windows, use Makefile as is. For UNIX, comment out lines 26, 27 and uncomment 28, 29. Changed file extension of ast.mli to ast.ml to mimic MicroC and so that the Makefile would work. Please remove ast.mli from src.	Dec 16 (5 days ago)	Monica
r57	cleaned up more error messages	Dec 09, 2010	Jiaying
r56	more cleanup of interpreter, better error messages	Dec 09, 2010	Jiaying
r55	cleaned up interpreter, minor bug fixes	Dec 09, 2010	Jiaying

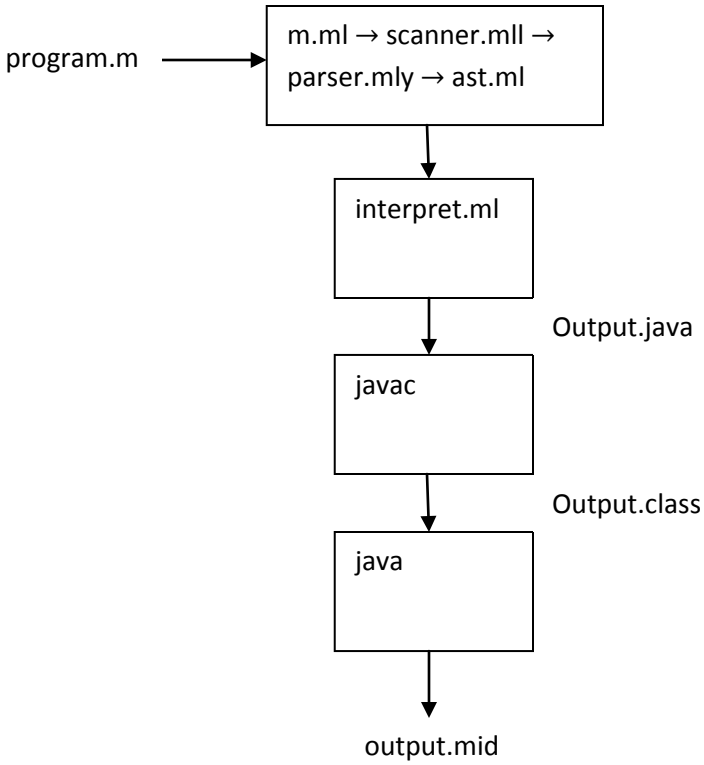
Rev	Commit log message	Date	Author
r54	Fixed if/for/while comparison bug	Dec 09, 2010	Jiaying
r53	Added comparison operator support for more types	Dec 09, 2010	Jiaying
r52	Fixed bugs in interpreter, added first m test file, created m driver program.	Dec 08, 2010	Jiaying
r51	Finished implementing add in interpreter	Dec 08, 2010	Jiaying
r50	interpreter improvements	Dec 08, 2010	Jiaying
r49	interpreter - removed unnecessary comments	Dec 08, 2010	Jiaying
r48	Made changes to ast and parser to allow assignment variant operators like +=, ++, etc	Dec 08, 2010	Jiaying
r47	Finished assignment in interpreter	Dec 08, 2010	Jiaying
r46	Put ++/-- back in for now. Interpreter - Changed BinOp, MemberAccess to reflect parser. In the middle of implementing AssignOp.	Dec 08, 2010	Jiaying
r45	removed ++ and -- from the ast/parser/scanner	Dec 08, 2010	Jiaying
r44	Updated parser and ast with type changes. Interpreter now compiles!	Dec 07, 2010	Jiaying

Rev	Commit log message	Date	Author
r43	Slight change in the interpreter	Dec 07, 2010	Jiaying
r42	re-implemented ++ and --	Dec 07, 2010	Monica
r41	deleted redundant function declaration grammar rule for void type	Dec 07, 2010	Monica
r40	IT COMPILES!!!	Dec 07, 2010	Monica
r39	intexpr, floatexpr, boolexpr, ... collapsed into just expr Compile error remains: type string used with type Ast.expr	Dec 07, 2010	Monica
r38	interpreter - function variable binding should work now	Dec 07, 2010	Jiaying
r37	Working on function variable binding	Dec 07, 2010	Jiaying
r36	interpreter - memberaccess and add works now	Dec 06, 2010	Jiaying
r35	improved interpreter	Dec 06, 2010	Jiaying
r34	fixed some issues that I had forgot to commit earlier	Dec 06, 2010	Yiling
r33	removed variable definition and declaration in the same statement.	Dec 06, 2010	Yiling
r32	assign functions in parser consolidated as in BinOp through the use of Assign ast modified to include Assign and other new functions, assignment operators are defined as assign type elseif removed	Dec 05, 2010	Monica
r31	Interpreter eval function now returns mtype instead of string	Dec 05, 2010	Jiaying
r30	Changed Not to take expr	Dec 05, 2010	Jiaying
r29	corrected naming conventions	Dec 05, 2010	Monica
r28	grammar for expressions polished	Dec 05, 2010	Monica
r27	member access and identifiers dealt with differently for easier development of interpreter	Dec 05, 2010	Monica
r26	Changed interpret.ml so that eval returns same time - string	Dec 04, 2010	Jiaying
r25	Restructured folder system so integration with subclipse works.	Dec 04, 2010	Jiaying
r24	[No log message]	Dec 03, 2010	Monica
r23	floats now accept 0.5 but not .5, modified in scanner.cmi parser's expressions drastically modified: 5++ no longer valid, accept variables (x) and attributes (note.pitch) as lvalues and rvalues correctly some commenting done for clarity fixed bug: (1 + 2) + 3 did not work previously, works now compile error persists: "parser.mly line 85 expression has type string but is used with type Ast.expr"	Dec 03, 2010	Monica
r22	Removed array functionality Modified idexpr and expr - member access and call functionality now only works for idexprs and are considered exprs	Dec 03, 2010	Monica
r21	Removed array functionality Corrected scanner's single	Dec 03, 2010	Monica

	comment recognition Created pitchexpr to allow pitch assignment Modified idexpr and expr - member access and call functionality now only works for idexprs		
r20	deleted unnecessary object files	Dec 03, 2010	Jiaying
r19	Added GetElement and MemberAccess to ast.mli and interpret.ml	Dec 02, 2010	Jiaying
r18	Implemented most of the expression evaluations in interpret.ml. Made changes to parser.mly and ast.mli for consistency purposes.	Dec 02, 2010	Jiaying
r17	changed types to match parser	Nov 23, 2010	Jiaying
r16	Fixed syntax error	Nov 23, 2010	Jiaying
r15	Pitchliteral will always be interpreted as an int. Fixed array access bug.	Nov 23, 2010	Jiaying
r14	changed ModInt -> Mod	Nov 23, 2010	Jiaying
r13	added more binops	Nov 23, 2010	Jiaying
r12	expr separated out into intexpr, floatexpr, boolexpr, etc. This resolves type ambiguity for interpreter. Removed unary + Possible issues: precedence	Nov 23, 2010	Monica
r11	[No log message]	Nov 23, 2010	Yiling
r10	Interpreter has been added	Nov 14, 2010	Jiaying
r9	[No log message]	Nov 14, 2010	Yiling
r8	ast and parser updated	Nov 14, 2010	Yiling
r7	compile.ml from MicroC. No syntax errors, but need to update to work with our ast.	Nov 14, 2010	Monica
r6	Deleted erroniously created directory "compile.ml" containing a duplicated m-music-language	Nov 14, 2010	Monica
r5	parser updated	Nov 14, 2010	Yiling

Rev	Commit log message	Date	Author
r4	test commit	Nov 14, 2010	Yiling
r3	Added compile.ml from MicroC. Compile error: File "compile.ml", line 48, characters 5-14: Error: Unbound constructor Literal Needs to be updated to work with our ast.	Nov 14, 2010	Monica
r2	Initial import	Nov 14, 2010	Yiling
r1	Initial directory structure.	Nov 14, 2010	---

5. Architectural Design



The major interface we used was between the interpreting stage and outputting the Java source file. This interface was more of a verbal agreement than code. Basically, we discussed a standard way to obtain each of the notes stored in order to write them to MIDI. To that end, we structured the code to be able to iterate through the data structures of notes stored in memory and then convert them easily into Java code. In order to support this on the Java side, helper methods needed to be written to facilitate the transfer of information from the O'Caml code to Java code. Thus, the final Java code consisted of all the pre-build methods that were simply stored as a string and written by the compiler into a new Java file, and each individual note was converted into a three-line block of Java code specifically designed to add the note's data into the final file

The scanner, parser, and ast were primarily implemented by Monica Ramirez-Santana. Their output was passed to the interpreter, which was primarily implemented by Jiaying Xu. The portion of the interpreter that deals with writing the Java source code was primarily implemented by Yiling Hu.

6. Test Plan

6.1 Representative Programs

See Appendix D for three representative programs written in the m language. The corresponding Java source programs for each are provided.

6.2 Test Suites

The most fundamental capabilities of our language are arithmetic manipulation, derived types which represent musical concepts, and control structures and user-defined functions for algorithmic composition. For that reason, these things and related concepts are the most heavily tested. Other features also tested include global variables and recursion.

Testing occurs for the most part at the stage of the interpreter. This is sufficient because at this stage, all computation for arithmetic manipulation has been done, values for derived types have been computed, and control structures and functions are completely executed.

The only capability that cannot be tested in this way is the correctness of the sounds produced at the MIDI stage. The only way to test this is with the human ear – the representative programs in 6.1 and the test program described in 6.2.2 were fully compiled and the output.mid files were played with Windows media player. The resulting sounds were deemed to be correct by our team's musicians, Yiling Hu and Jiaying Xu.

Test suites were developed and testing was primarily done by Monica Ramirez-Santana.

6.2.1 Arithmetic

Arithmetic

The following tests are performed in the test_arith.m file.

Test	Motivation	Expected Result	Result
print(1 + 2);	Integer addition	3	3
print(1 - 2);	Integer subtraction	-1	-1
print(2 * 2);	Integer multiplication	4	4
print(1 / 2);	Integer division	0	0
print(5 % 2);	Modulus	1	1
print(1.0 + 2.0);	Floating point addition	3.	3.
print(1.0 - 2.0);	Floating point subtraction	-1.	-1.
print(2.0 * 2.0);	Floating point multiplication	4.	4.
print(1.0 / 2.0);	Floating point division	0.5	0.5
print(1 + 2 * 3 + 4);	Order of operations	11	11
print((1 + 2) * (3 + 4));	Precedence/order of operations	21	21

Assignment

Assignment is an operation crucial to arithmetic manipulation with objects. The following tests are performed in the test_assignment.m file.

Test	Motivation	Expected Result	Result
print(x=1); print(x);	Assign literal to variable Evaluation of assignment Assignment functionality	 1 1	 1 1
print(n.pitch = 42); print(n.pitch);	Assign literal to member Evaluation of assignment Assignment functionality	 42 42	 42 42
print(a=x);	Assignment of variable	1	1
print(b = n.pitch);	Assignment of member	42	42

print(b = a = n.pitch = x = 13);
print(a);
print(n.pitch);
print(x);
	Successive assignment	13
		13
		13
		13
			13
			13
			13
			13

print(x += 1);	+=	14	14
print(x++);	++	15	15
print(x -= 1);	-=	14	14
print(x--);	--	13	13
print(x *= 10);	*=	130	130
print(x /= 2);	/=	65	65
print(x %= 2);	%=	1	1
print(x = x);	Self assignment	1	1

6.2.2 Derived types

The following tests are performed in hello_world.m.

Test	Motivation	Expected Result	Result
n.pitch = C4;	Pitch field of note	Note played is middle C	Success
n.duration = 1.0;	Duration field of note	Note played for 4 seconds	Success
n.intensity = 100;	Intensity field of note	Note is audible and fairly loud	Success
s.instrument = 0;	Instrument field of staff	Section of song played on the piano	Success
p.bpm = 60;	BPM field of part	Note played for 4 seconds	Success
p.beatsig = 0.25;	Beat signature field of part	Note played for 4 seconds	Success
add(c, n);	Add note to chord	Song plays	Success
add(s, c);	Add chord to staff	Song plays	Success
add(p, s);	Add staff to part	Song plays	Success
play(p);	Play invocation	Song written to MIDI file	Success

6.2.3 Control Structures

Logic

Logical operators are necessary for control structures. The following tests are performed in test_logic.m.

Test	Motivation	Expected Result	Result

print(!true); print(!false);	!	false true	false true

print(true && true);
print(true && false);
print(false && true);
print(false && false);
	&&	true
		false
		false
		false
			true
			false
			false
			false

print(true || true);
print(true || false);
print(false || true);
print(false || false);
	||	true
		true
		true
		false
			true
			true
			true
			false

print(1 == 2); print(1 == 1);	==	false true	false true
print(1 != 2); print(1 != 1);	!=	true false	true false
print(1 < 2); print(2 < 1);	<	true false	true false

print(1 <= 2);
print(1 <= 1);
print(2 <= 1);
	<=	true
		true
		false
			true
			true
			false

print(1 > 2); print(2 > 1);	>	false true	false true

print(1 >= 2);
print(1 >= 1);
print(2 >= 1);
	>=	false
		true
		true
			false
			true
			true

print(!true || true);	Order of operations	true	true
print(!(true || true));	Precedence/order of operations	false	false

if

The following test is performed in test_if.m.

Test	**Motivation**	**Expected Result**	**Result**

void main()
{
if(true)
{
print(0);
}
 	
if(false)
{
print(1);
}
else
{
print(2);
}
}
	if and if/else construction
	Correct functionality
		0
		2
			0
			2

for

The following test is performed in test_for.m.

Test	Motivation	Expected Result	Result

void main()
{
int i;
 	
for(i = 0; i < 10; i++)
{
print(i);
}
}
	for(expression$_{opt}$; expression$_{opt}$; expression$_{opt}$) compound-statement
	construction
	Correct functionality
		0
		1
		2
		3
		4
		5
		6
		7
		8
		9
			0
			1
			2
			3
			4
			5
			6
			7
			8
			9

while

The following test is performed in test_while.m.

Test	Motivation	Expected Result	Result

void main()
{
int i;
i = 0;
 	
while(i < 10)
{
print(i);
i++;
}
}
	while(expression) compound-statement
	construction
	Correct functionality
		0
		1
		2
		3
		4
		5
		6
		7
		8
		9
			0
			1
			2
			3
			4
			5
			6
			7
			8
			9

6.2.4 Functions

The following test is performed in test_func.m.

Test	Motivation	Expected Result	Result

int const( )
{
return 0;
}
 	
void prnt(int a, int b)
{
print(a);
print(b);
}
	type identifier(parameter-list$_{opt}$) { declaration-list$_{opt}$ statement-list$_{opt}$ }
	construction
	Correct functionality
		0
		1
		2
		3
			0
			1
			2
			3

int plus(int a, int b)
{
return a + b;
}
 	
void main()
{
print(const());
prnt(1, 2);
print(plus(1, 2));
}			

6.2.5 Other Features

Global Variables

The following test is performed in test_global.m.

Test	Motivation	Expected Result	Result

int a;
int b;
 	
int printa()
{
print(a);
}
 	
int printb()
{
print(b);
}
 	
int increment_globals()
{
a++;
b++;
}
 	
void main()
{
a = 42;
b = 21;
printa();
printb();
increment_globals();
printa();
printb();
}
	Global variables	42
		21
		43
		22
			42
			21
			43
			22

Recursion

The following test is performed in test_fib.m.

Test	Motivation	Expected Result	Result

int fib(int x)
{
if (x < 2)
{
return 1;
}
 	
return fib(x-1) + fib(x-2);
}
 	
void main()
{
print(fib(0));
print(fib(1));
print(fib(2));
print(fib(3));
print(fib(4));
print(fib(5));
}
	Recursion	1
		1
		3
		5
		8
			1
			1
			3
			5
			8

7. Lessons Learned

7.1 Yiling Hu

This project taught me how to work in a group on a fairly large software system. I realized it's important to have a "project manager" to schedule meetings and make sure everyone is doing their part of the project. Also, it makes sense to divide up the work based on each team member's strengths, as usually that ends up with the best results in the fastest time. For parts of the project that were new to all members of the team, it was helpful to have all members of the team present working on the same code in an agile fashion to quickly find errors. For more individual coding assignments, having a version control made it much easier to send and receive the latest code from other team members. Finally, actually taking the time early on to plan and learn O'caml helps tremendously when the time comes to finally write the code.

7.2 Monica Ramirez-Santana

This project gave me a lot more experience in group work and larger-scale software development. I learned that it was very helpful for someone to assume the role of project manager. I also learned it was essential to save time by having one person take the lead in providing direction and vision for the software. Oftentimes, we spent quite a bit of time discussing whether one feature or another would be useful and the implications of implementing/not implementing it. With just one person calling most of the shots on design decisions, this time could have been spent differently. It was also important to have each person perform the roles that they would be most suited for and were most interested in. And finally, I learned that extensive documentation of all group discussions and design decisions would have been a huge time-saver. As for advice to future students: start early, stick to a regular meeting schedule throughout the semester, and document everything!

7.3 Jiaying Xu

After completing this project, I feel that I have gained a fairly good grasp of O'Caml and functional programming. In terms of the group and project management aspect, I learned that it's important to solidify the concept and exactly what the specifications are early on in the process. We tried to meet regularly and work together whenever possible. It's a lot easier to discuss things in person as a group instead of online. When dividing up the project, we made sure to assign things to each person that it was something they wanted to do and played well with their individual strengths. Setting up a source control system was also essential. In terms of advice to future teams, you should make sure you start early. Even though it seems like you have the entire semester to the project, time goes by fast and leaving the project for the last minute isn't a good idea when you haven't written a compiler or in O'Caml before. It's important to have deadlines set that the team will actually follow and make sure everyone is on the same page about the work being done.

Appendix A: Grammar

literal:
- *integer-literal*
- *floating-literal*
- *boolean-literal*
- *pitch-literal*

type:
- *void*
- *int*
- *float*
- *bool*
- *note*
- *chord*
- *staff*
- *part*

expression:
- *identifier*
- *literal*
- *expression member-access-op expression*
- *expression assignment-op expression*
- *expression binary-op expression*
- *prefix-op expression*
- *expression postfix-op*
- *(expression)*

member-access-op:
- .

postfix-op: one of
- ++ -- *(argument-expression-list$_{opt}$)*

argument-expression-list:
- *expression*
- *argument-expression-list , expression*

prefix-operator:
- *!*

binary-op:
- *arithmetic-op*
- *relational-op*
- *logical-op*

arithmetic-op: one of
`*  /  %  +  -`

relational-op: one of
`<  >  <=  >=  ==  !=`

logical-op: one of
`&&  ||`

assignment-op: one of
`=  *=  /=  %=  +=  -=`

declaration:
type identifier;

statement:
expression;
compound-statement
selection-statement
iteration-statement
return $expression_{opt}$*;*

compound-statement:
{ $statement\text{-}list_{opt}$ *}*

statement-list:
statement
statement-list statement

selection-statement:
if(expression) compound-statement
if(expression) compound-statement else compound-statement

iteration-statement:
while(expression) compound-statement
for($expression_{opt}$*;* $expression_{opt}$*;* $expression_{opt}$*) compound-statement*

function-definition:
type identifier($parameter\text{-}list_{opt}$*) { declaration-*$list_{opt}$ $statement\text{-}list_{opt}$ *}*

parameter-list:
type identifier
parameter-list, type identifier

declaration-list:
    declaration
    declaration-list declaration

program:

    declaration-list$_{opt}$ *function-definition-list*$_{opt}$

function-definition-list:
    function-definition
    function-definition-list function-definition

Appendix B: Standard Library

B.1 add()

add(type1 a, type2 b);

The add standard library function adds b to the collection of type2 contained in a. type 1 can be `chord`, `staff`, or `part`. If type1 is `chord`, type2 must be `part`; if it is `staff`, type2 must be `note` or `chord`; if it is `part`, type2 must be `staff`.

B.2 play()

play(part p);

The play standard library function outputs the argument to the output.mid file. If play is called more than once in a program, the Output.java is rewritten every time. This means that only the last invocation of play has any effect on the output.mid file.

B.3 print()

print(type t);

The print standard library function prints p to stdout. type can be any type, but will behave differently for different types. If type is `int`, `float`, or `bool`, print will output the value of t. If type is anything else, print will output the type.

B.4 randint()

randint(int i);

Returns a pseudorandom integer in the range [0, i).

B.4 randfloat()

randfloat(float f);

Returns a pseudorandom float in the range [0, f).

Appendix C: m Pitch and Instrument Values

A mapping of notes to their numeric values can be found below.

Octave	Note Numbers											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

The following is a list of all the instruments we are supporting. The number next to them is the corresponding MIDI number.

Piano

1 Acoustic Grand Piano
2 Bright Acoustic Piano
3 Electric Grand Piano
4 Honky-tonk Piano
5 Electric Piano 1
6 Electric Piano 2
7 Harpsichord
8 Clavinet

Chromatic Percussion

9 Celesta
10 Glockenspiel
11 Music Box
12 Vibraphone
13 Marimba
14 Xylophone
15 Tubular Bells
16 Dulcimer

Organ

17 Drawbar Organ
18 Percussive Organ
19 Rock Organ
20 Church Organ
21 Reed Organ
22 Accordion
23 Harmonica
24 Tango Accordion

Guitar

25 Acoustic Guitar (nylon)
26 Acoustic Guitar (steel)
27 Electric Guitar (jazz)
28 Electric Guitar (clean)
29 Electric Guitar (muted)
30 Overdriven Guitar
31 Distortion Guitar
32 Guitar Harmonics

Bass

33 Acoustic Bass
34 Electric Bass (finger)
35 Electric Bass (pick)
36 Fretless Bass
37 Slap Bass 1
38 Slap Bass 2
39 Synth Bass 1
40 Synth Bass 2

Strings

41 Violin

42 Viola
43 Cello
44 Contrabass
45 Tremolo Strings
46 Pizzicato Strings
47 Orchestral Harp
48 Timpani

Ensemble

49 String Ensemble 1
50 String Ensemble 2
51 Synth Strings 1
52 Synth Strings 2
53 Choir Aahs
54 Voice Oohs
55 Synth Choir
56 Orchestra Hit

Brass

57 Trumpet
58 Trombone
59 Tuba
60 Muted Trumpet
61 French Horn
62 Brass Section
63 Synth Brass 1
64 Synth Brass 2

Reed

65 Soprano Sax
66 Alto Sax
67 Tenor Sax
68 Baritone Sax
69 Oboe
70 English Horn
71 Bassoon
72 Clarinet

Pipe

73 Piccolo
74 Flute
75 Recorder
76 Pan Flute
77 Blown Bottle

78 Shakuhachi
79 Whistle
80 Ocarina

Synth Lead

81 Lead 1 (square)
82 Lead 2 (sawtooth)
83 Lead 3 (calliope)
84 Lead 4 (chiff)
85 Lead 5 (charang)
86 Lead 6 (voice)
87 Lead 7 (fifths)
88 Lead 8 (bass + lead)

Synth Pad

89 Pad 1 (new age)
90 Pad 2 (warm)
91 Pad 3 (polysynth)
92 Pad 4 (choir)
93 Pad 5 (bowed)
94 Pad 6 (metallic)
95 Pad 7 (halo)
96 Pad 8 (sweep)

Synth Effects

97 FX 1 (rain)
98 FX 2 (soundtrack)
99 FX 3 (crystal)
100 FX 4 (atmosphere)
101 FX 5 (brightness)
102 FX 6 (goblins)
103 FX 7 (echoes)
104 FX 8 (sci-fi)

Ethnic

105 Sitar
106 Banjo
107 Shamisen
108 Koto
109 Kalimba
110 Bagpipe
111 Fiddle
112 Shanai

Percussive

113 Tinkle Bell
114 Agogo
115 Steel Drums
116 Woodblock
117 Taiko Drum
118 Melodic Tom
119 Synth Drum

Sound effects

120 Reverse Cymbal
121 Guitar Fret Noise
122 Breath Noise
123 Seashore
124 Bird Tweet
125 Telephone Ring
126 Helicopter
127 Applause
128 Gunshot

Appendix D: Representative Programs

D.1 Twinkle, Twinkle, Little Star

The following example illustrates that the m language has the full capability of regular musical composition. Note that twinkle.m has 73 lines of code when compared to Output.java's 344 lines.

twinkle.m

```
note makeNote(int x, float y, int z)
{
	note N;
	N.pitch = x;
	N.duration = y;
	N.intensity = z;

	return N;
}
void main()
{
	part P;
	staff S;
	chord C;
	chord G;
	chord A;
	chord F;
	chord E;
```

```
	chord D;
	note c;
	note g;
	note f;
	note e;
	note d;
	note a;
	note longg;
	note longc;
	chord longA;
	chord longC;

	P.bpm = 100;
	P.beatsig = 0.25;

	S.instrument = 0;


	c = makeNote(C4, 0.25, 100);
	g = makeNote(G4, 0.25, 100);
	a = makeNote(A4, 0.25, 100);
	f = makeNote(F4, 0.25, 100);
	e = makeNote(E4, 0.25, 100);
	d = makeNote(D4, 0.25, 100);

	add(C , c);
	add(G , g);
	add(A , a);

	longg = makeNote(G4, 0.5, 100);
	longc = makeNote(C4, 0.5, 100);

	add(S, c);
	add(S, c);
	add(S, g);
	add(S, g);
	add(S, a);
	add(S, a);

	add(S, longg);
	add(S, f);
	add(S, f);
	add(S, e);
	add(S, e);
	add(S, d);
	add(S, d);
	add(S, longc);

	add(P, S);

	play(P);
}
```

Output.java

```
import java.io.File;

import java.io.IOException;

import java.math.BigInteger;

import java.util.LinkedList;

import java.util.List;

import javax.sound.midi.InvalidMidiDataException;

import javax.sound.midi.MetaMessage;

import javax.sound.midi.MidiEvent;

import javax.sound.midi.MidiSystem;

import javax.sound.midi.Sequence;

import javax.sound.midi.ShortMessage;

import javax.sound.midi.Track;


public class Output

{

	Sequence sequence = null;

	List<Track> trackList;

	int tempo;

	double beatSig;

	int beatTicks;


	public Output(int bpm, double beatsig)

	{

		this.tempo = bpm;
```

```
this.beatSig = beatsig;
beatTicks = (int) (beatsig/(1.0/32.0));
trackList = new LinkedList<Track>();

try
{
        sequence = new Sequence(Sequence.PPQ, beatTicks);
}
catch (InvalidMidiDataException e)
{
        e.printStackTrace();
        System.exit(1);
}

makeTrack();

try
{
        MidiSystem.write(sequence, 1, new File("output.mid"));
        System.out.println("file written");
}
catch (IOException e)
{
        e.printStackTrace();
        System.exit(1);
}
```

```
}

private byte[] getTempo(int bpm)
{
	int ms = 60000000;
	Integer i = new Integer(ms/bpm);
	BigInteger bi = new BigInteger(i.toString());
	return bi.toByteArray();
}

private Track createTrack(int instrument)
{
	Track t = sequence.createTrack();
	try
	{
		MetaMessage mt = new MetaMessage();
		byte[] bt = getTempo(tempo);
		mt.setMessage(0x51 ,bt, bt.length);
		MidiEvent me = new MidiEvent(mt, 0);
		t.add(me);

		ShortMessage sm = new ShortMessage();
		sm.setMessage(192, instrument, 0);
		me = new MidiEvent(sm, 0);
		t.add(me);
	} catch (Exception e)
```

```
            {
                    e.printStackTrace();
            }
            return t;
    }

    private MidiEvent createNoteOnEvent(int pitch, long lTick, int intensity)
    {
            return createNoteEvent(ShortMessage.NOTE_ON, pitch, intensity, lTick);
    }

    private MidiEvent createNoteOffEvent(int pitch, long lTick)
    {
            return createNoteEvent(ShortMessage.NOTE_OFF, pitch, 0, lTick);
    }

    private MidiEvent createNoteEvent(int nCommand, int nKey, int intensity, long lTick)
    {
            ShortMessage message = new ShortMessage();
            try
            {
                    message.setMessage(nCommand, 0,          // always on channel 1
                                                             nKey,
                                                             intensity);
            }
            catch (InvalidMidiDataException e)
```

```
            {
                e.printStackTrace();
                System.exit(1);
            }
            MidiEvent event = new MidiEvent(message,
                                                                    lTick);
            return event;
    }

    private long getDuration(double duration)
    {
            long d = Math.round((duration/beatSig) * beatTicks);
            return d;
    }

    private List<MidiEvent> addNote(int startTick, int pitch, long lTick, int intensity)
    {
            List<MidiEvent> list = new LinkedList<MidiEvent>();
            System.out.println(startTick);
            System.out.println(startTick + lTick);

            list.add(createNoteOnEvent( pitch, intensity, startTick));
            list.add(createNoteOffEvent(pitch, startTick + lTick));

            return list;
    }
```

```
        private void makeTrack()
        {
                int tick;
                Track t;

t = createTrack(0);
tick = 0;

t.add(createNoteOnEvent(60, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(60, tick));

t.add(createNoteOnEvent(60, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(60, tick));

t.add(createNoteOnEvent(67, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(67, tick));

t.add(createNoteOnEvent(67, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(67, tick));

t.add(createNoteOnEvent(69, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(69, tick));

t.add(createNoteOnEvent(69, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(69, tick));

t.add(createNoteOnEvent(67, tick,100));
tick += getDuration(0.5);
t.add(createNoteOffEvent(67, tick));

t.add(createNoteOnEvent(65, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(65, tick));

t.add(createNoteOnEvent(65, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(65, tick));

t.add(createNoteOnEvent(64, tick,100));
tick += getDuration(0.25);
```

```
t.add(createNoteOffEvent(64, tick));

t.add(createNoteOnEvent(64, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(64, tick));

t.add(createNoteOnEvent(62, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(62, tick));

t.add(createNoteOnEvent(62, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(62, tick));

t.add(createNoteOnEvent(60, tick,100));
tick += getDuration(0.5);
t.add(createNoteOffEvent(60, tick));


        }



public static void main(String[] args)

        {

new Output(100, 0.25);

        }



}
```

D.2 Chromatic Scale

The following example illustrates how basic algorithmic composition works in the m language. Note that chromaticscale.m has 31 lines of code as opposed to Output.java's 340 lines.

chromaticscale.m

```
note makeNote(int x, float y, int z)
{
        note N;
        N.pitch = x;
        N.duration = y;
        N.intensity = z;

        return N;
}
```

```
void main()
{
	part P;
	staff S;
	int i;
	note temp;

	for (i = 0; i< 14; i++)
	{
		temp = makeNote(C4+i, 0.25, 100);
		add(S, temp);
	}

	P.bpm = 100;
	P.beatsig = 0.25;

	add(P, S);

	play(P);
}
```

Output.java

```
import java.io.File;

import java.io.IOException;

import java.math.BigInteger;

import java.util.LinkedList;

import java.util.List;

import javax.sound.midi.InvalidMidiDataException;

import javax.sound.midi.MetaMessage;

import javax.sound.midi.MidiEvent;

import javax.sound.midi.MidiSystem;

import javax.sound.midi.Sequence;

import javax.sound.midi.ShortMessage;

import javax.sound.midi.Track;
```

```
public class Output
{
	Sequence sequence = null;
	List<Track> trackList;
	int tempo;
	double beatSig;
	int beatTicks;

	public Output(int bpm, double beatsig)
	{
		this.tempo = bpm;
		this.beatSig = beatsig;
		beatTicks = (int) (beatsig/(1.0/32.0));
		trackList = new LinkedList<Track>();

		try
		{
			sequence = new Sequence(Sequence.PPQ, beatTicks);
		}
		catch (InvalidMidiDataException e)
		{
			e.printStackTrace();
			System.exit(1);
		}
```

```
		makeTrack();


		try
		{
			MidiSystem.write(sequence, 1, new File("output.mid"));
			System.out.println("file written");
		}
		catch (IOException e)
		{
			e.printStackTrace();
			System.exit(1);
		}
	}


	private byte[] getTempo(int bpm)
	{
		int ms = 60000000;
		Integer i = new Integer(ms/bpm);
		BigInteger bi = new BigInteger(i.toString());
		return bi.toByteArray();
	}


	private Track createTrack(int instrument)
	{
		Track t = sequence.createTrack();
		try
```

```
            {
                MetaMessage mt = new MetaMessage();
                byte[] bt = getTempo(tempo);
                mt.setMessage(0x51 ,bt, bt.length);
                MidiEvent me = new MidiEvent(mt, 0);
                t.add(me);

                ShortMessage sm = new ShortMessage();
                sm.setMessage(192, instrument, 0);
                me = new MidiEvent(sm, 0);
                t.add(me);
            } catch (Exception e)
            {
                e.printStackTrace();
            }
            return t;
    }

    private MidiEvent createNoteOnEvent(int pitch, long lTick, int intensity)
    {
            return createNoteEvent(ShortMessage.NOTE_ON, pitch, intensity, lTick);
    }

    private MidiEvent createNoteOffEvent(int pitch, long lTick)
    {
            return createNoteEvent(ShortMessage.NOTE_OFF, pitch, 0, lTick);
```

```
}


private MidiEvent createNoteEvent(int nCommand, int nKey, int intensity, long lTick)
{
        ShortMessage message = new ShortMessage();
        try
        {
                message.setMessage(nCommand, 0,         // always on channel 1
                                                        nKey,
                                                        intensity);
        }
        catch (InvalidMidiDataException e)
        {
                e.printStackTrace();
                System.exit(1);
        }
        MidiEvent       event = new MidiEvent(message,
                                                                        lTick);
        return event;
}


private long getDuration(double duration)
{
        long d = Math.round((duration/beatSig) * beatTicks);
        return d;
}
```

```
	private List<MidiEvent> addNote(int startTick, int pitch, long lTick, int intensity)
	{
		List<MidiEvent> list = new LinkedList<MidiEvent>();
		System.out.println(startTick);
		System.out.println(startTick + lTick);

		list.add(createNoteOnEvent( pitch, intensity, startTick));
		list.add(createNoteOffEvent(pitch, startTick + lTick));

		return list;
	}
	private void makeTrack()
	{
		int tick;
		Track t;

t = createTrack(0);
tick = 0;

t.add(createNoteOnEvent(60, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(60, tick));

t.add(createNoteOnEvent(61, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(61, tick));

t.add(createNoteOnEvent(62, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(62, tick));

t.add(createNoteOnEvent(63, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(63, tick));
```

```
t.add(createNoteOnEvent(64, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(64, tick));

t.add(createNoteOnEvent(65, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(65, tick));

t.add(createNoteOnEvent(66, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(66, tick));

t.add(createNoteOnEvent(67, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(67, tick));

t.add(createNoteOnEvent(68, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(68, tick));

t.add(createNoteOnEvent(69, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(69, tick));

t.add(createNoteOnEvent(70, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(70, tick));

t.add(createNoteOnEvent(71, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(71, tick));

t.add(createNoteOnEvent(72, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(72, tick));

t.add(createNoteOnEvent(73, tick,100));
tick += getDuration(0.25);
t.add(createNoteOffEvent(73, tick));
        }

public static void main(String[] args)

        {

new Output(100, 0.25);

        }

}
```

D.3 Chromatic Fifths

Same as chromaticscale.m but with chords.

chromaticfifths.m

```
note makeNote(int x, float y, int z)
{
	note N;
	N.pitch = x;
	N.duration = y;
	N.intensity = z;

	return N;
}

chord makeChord(note n1, note n2)
{
	chord C;
	add(C, n1);
	add(C, n2);
	return C;
}

void main()
{
	part P;
	staff S;
	chord C;
	int i;
	note temp;
	note tempfifth;

	for (i = 0; i< 14; i++)
	{
		temp = makeNote(C4+i, 0.25, 100);
		tempfifth = makeNote(G4+i, 0.25, 100);

		C = makeChord(temp, tempfifth);
		add(S, C);
	}

	P.bpm = 100;
	P.beatsig = 0.25;

	add(P, S);

	play(P);
}
```

Output.java

```
import java.io.File;
import java.io.IOException;
import java.math.BigInteger;
import java.util.LinkedList;
import java.util.List;
import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.MetaMessage;
import javax.sound.midi.MidiEvent;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.Sequence;
import javax.sound.midi.ShortMessage;
import javax.sound.midi.Track;

public class Output
{
	Sequence sequence = null;
	List<Track> trackList;
	int tempo;
	double beatSig;
	int beatTicks;

	public Output(int bpm, double beatsig)
	{
```

```
this.tempo = bpm;

this.beatSig = beatsig;

beatTicks = (int) (beatsig/(1.0/32.0));

trackList = new LinkedList<Track>();



try

{

        sequence = new Sequence(Sequence.PPQ, beatTicks);

}

catch (InvalidMidiDataException e)

{

        e.printStackTrace();

        System.exit(1);

}



makeTrack();



try

{

        MidiSystem.write(sequence, 1, new File("output.mid"));

        System.out.println("file written");

}

catch (IOException e)

{

        e.printStackTrace();

        System.exit(1);
```

```
            }
        }


        private byte[] getTempo(int bpm)
        {
            int ms = 60000000;
            Integer i = new Integer(ms/bpm);
            BigInteger bi = new BigInteger(i.toString());
            return bi.toByteArray();
        }


        private Track createTrack(int instrument)
        {
            Track t = sequence.createTrack();
            try
            {
                MetaMessage mt = new MetaMessage();
                byte[] bt = getTempo(tempo);
                mt.setMessage(0x51 ,bt, bt.length);
                MidiEvent me = new MidiEvent(mt, 0);
                t.add(me);


                ShortMessage sm = new ShortMessage();
                sm.setMessage(192, instrument, 0);
                me = new MidiEvent(sm, 0);
                t.add(me);
```

```
            } catch (Exception e)
            {
                  e.printStackTrace();
            }
            return t;
      }

      private MidiEvent createNoteOnEvent(int pitch, long lTick, int intensity)
      {
            return createNoteEvent(ShortMessage.NOTE_ON, pitch, intensity, lTick);
      }

      private MidiEvent createNoteOffEvent(int pitch, long lTick)
      {
            return createNoteEvent(ShortMessage.NOTE_OFF, pitch, 0, lTick);
      }

      private MidiEvent createNoteEvent(int nCommand, int nKey, int intensity, long lTick)
      {
            ShortMessage message = new ShortMessage();
            try
            {
                  message.setMessage(nCommand, 0,          // always on channel 1
                                                      nKey,
                                                      intensity);
            }
```

```
            catch (InvalidMidiDataException e)
            {
                  e.printStackTrace();
                  System.exit(1);
            }
            MidiEvent           event = new MidiEvent(message,
                                                                                    lTick);
            return event;
      }

      private long getDuration(double duration)
      {
            long d = Math.round((duration/beatSig) * beatTicks);
            return d;
      }

      private List<MidiEvent> addNote(int startTick, int pitch, long lTick, int intensity)
      {
            List<MidiEvent> list = new LinkedList<MidiEvent>();
            System.out.println(startTick);
            System.out.println(startTick + lTick);

            list.add(createNoteOnEvent( pitch, intensity, startTick));
            list.add(createNoteOffEvent(pitch, startTick + lTick));

            return list;
```

```
        }


        private void makeTrack()

        {

                int tick;

                Track t;


t = createTrack(0);
tick = 0;

t.add(createNoteOnEvent(67, tick,100));
t.add(createNoteOffEvent(67, tick + getDuration(0.25)));

t.add(createNoteOnEvent(60, tick,100));
t.add(createNoteOffEvent(60, tick + getDuration(0.25)));

tick += getDuration(0.25);

t.add(createNoteOnEvent(68, tick,100));
t.add(createNoteOffEvent(68, tick + getDuration(0.25)));

t.add(createNoteOnEvent(61, tick,100));
t.add(createNoteOffEvent(61, tick + getDuration(0.25)));

tick += getDuration(0.25);

t.add(createNoteOnEvent(69, tick,100));
t.add(createNoteOffEvent(69, tick + getDuration(0.25)));

t.add(createNoteOnEvent(62, tick,100));
t.add(createNoteOffEvent(62, tick + getDuration(0.25)));

tick += getDuration(0.25);

t.add(createNoteOnEvent(70, tick,100));
t.add(createNoteOffEvent(70, tick + getDuration(0.25)));

t.add(createNoteOnEvent(63, tick,100));
t.add(createNoteOffEvent(63, tick + getDuration(0.25)));

tick += getDuration(0.25);

t.add(createNoteOnEvent(71, tick,100));
t.add(createNoteOffEvent(71, tick + getDuration(0.25)));

t.add(createNoteOnEvent(64, tick,100));
```

```
t.add(createNoteOffEvent(64, tick + getDuration(0.25)));

tick += getDuration(0.25);

t.add(createNoteOnEvent(72, tick,100));
t.add(createNoteOffEvent(72, tick + getDuration(0.25)));

t.add(createNoteOnEvent(65, tick,100));
t.add(createNoteOffEvent(65, tick + getDuration(0.25)));

tick += getDuration(0.25);

t.add(createNoteOnEvent(73, tick,100));
t.add(createNoteOffEvent(73, tick + getDuration(0.25)));

t.add(createNoteOnEvent(66, tick,100));
t.add(createNoteOffEvent(66, tick + getDuration(0.25)));

tick += getDuration(0.25);

t.add(createNoteOnEvent(74, tick,100));
t.add(createNoteOffEvent(74, tick + getDuration(0.25)));

t.add(createNoteOnEvent(67, tick,100));
t.add(createNoteOffEvent(67, tick + getDuration(0.25)));

tick += getDuration(0.25);

t.add(createNoteOnEvent(75, tick,100));
t.add(createNoteOffEvent(75, tick + getDuration(0.25)));

t.add(createNoteOnEvent(68, tick,100));
t.add(createNoteOffEvent(68, tick + getDuration(0.25)));

tick += getDuration(0.25);

t.add(createNoteOnEvent(76, tick,100));
t.add(createNoteOffEvent(76, tick + getDuration(0.25)));

t.add(createNoteOnEvent(69, tick,100));
t.add(createNoteOffEvent(69, tick + getDuration(0.25)));

tick += getDuration(0.25);

t.add(createNoteOnEvent(77, tick,100));
t.add(createNoteOffEvent(77, tick + getDuration(0.25)));

t.add(createNoteOnEvent(70, tick,100));
t.add(createNoteOffEvent(70, tick + getDuration(0.25)));

tick += getDuration(0.25);
```

```
t.add(createNoteOnEvent(78, tick,100));
t.add(createNoteOffEvent(78, tick + getDuration(0.25)));

t.add(createNoteOnEvent(71, tick,100));
t.add(createNoteOffEvent(71, tick + getDuration(0.25)));

tick += getDuration(0.25);

t.add(createNoteOnEvent(79, tick,100));
t.add(createNoteOffEvent(79, tick + getDuration(0.25)));

t.add(createNoteOnEvent(72, tick,100));
t.add(createNoteOffEvent(72, tick + getDuration(0.25)));

tick += getDuration(0.25);

t.add(createNoteOnEvent(80, tick,100));
t.add(createNoteOffEvent(80, tick + getDuration(0.25)));

t.add(createNoteOnEvent(73, tick,100));
t.add(createNoteOffEvent(73, tick + getDuration(0.25)));

tick += getDuration(0.25);


        }



public static void main(String[] args)

        {

new Output(100, 0.25);

        }



}
```

D.4 Completely Randomly Generated Piano Piece

Notes were generated given a base note.

Randommusic.m

```
note makeNote(int x, float y, int z)
{
	note N;
	N.pitch = x;
	N.duration = y;
	N.intensity = z;

	return N;
}
chord makeChord(note n1, note n2)
{
	chord C;
	add(C, n1);
	add(C, n2);
	return C;
}
void main()
{
	part P;
	staff S;
	staff S1;
	int i;
	note temp1;
	note temp2;

	/* Randomly generates two notes
	with random length
	*/
	for (i = 0; i < 60; i++)
	{
		if (i<25 || i > 40)
		{
			temp1 = makeNote(C3 + randint(40), randfloat(1.0), 100);
			temp2 = makeNote(D4 + randint(30), randfloat(0.75), 100);
		}
		else
		{
			temp1 = makeNote(C3 + randint(40), randfloat(0.25), 100);
			temp2 = makeNote(D4 + randint(30), randfloat(0.125), 100);
		}

		add(S, temp1);
		add(S1, temp2);
	}

	S.instrument = 0;
	S1.instrument = 0;
```

```
        add(P, S);
        add(P, S1);

        play(P);
}
```

Output.java

```
import java.io.File;

import java.io.IOException;

import java.math.BigInteger;

import java.util.LinkedList;

import java.util.List;

import javax.sound.midi.InvalidMidiDataException;

import javax.sound.midi.MetaMessage;

import javax.sound.midi.MidiEvent;

import javax.sound.midi.MidiSystem;

import javax.sound.midi.Sequence;

import javax.sound.midi.ShortMessage;

import javax.sound.midi.Track;


public class Output

{

        Sequence sequence = null;

        List<Track> trackList;

        int tempo;

        double beatSig;

        int beatTicks;
```

```
public Output(int bpm, double beatsig)
{
	this.tempo = bpm;
	this.beatSig = beatsig;
	beatTicks = (int) (beatsig/(1.0/32.0));
	trackList = new LinkedList<Track>();

	try
	{
		sequence = new Sequence(Sequence.PPQ, beatTicks);
	}
	catch (InvalidMidiDataException e)
	{
		e.printStackTrace();
		System.exit(1);
	}

	makeTrack();

	try
	{
		MidiSystem.write(sequence, 1, new File("output.mid"));
		System.out.println("file written");
	}
	catch (IOException e)
```

```
		{
			e.printStackTrace();
			System.exit(1);
		}
	}

	private byte[] getTempo(int bpm)
	{
		int ms = 60000000;
		Integer i = new Integer(ms/bpm);
		BigInteger bi = new BigInteger(i.toString());
		return bi.toByteArray();
	}

	private Track createTrack(int instrument)
	{
		Track t = sequence.createTrack();
		try
		{
			MetaMessage mt = new MetaMessage();
			byte[] bt = getTempo(tempo);
			mt.setMessage(0x51 ,bt, bt.length);
			MidiEvent me = new MidiEvent(mt, 0);
			t.add(me);

			ShortMessage sm = new ShortMessage();
```

```
                    sm.setMessage(192, instrument, 0);
                    me = new MidiEvent(sm, 0);
                    t.add(me);
            } catch (Exception e)
            {
                    e.printStackTrace();
            }
            return t;
    }

    private MidiEvent createNoteOnEvent(int pitch, long lTick, int intensity)
    {
            return createNoteEvent(ShortMessage.NOTE_ON, pitch, intensity, lTick);
    }

    private MidiEvent createNoteOffEvent(int pitch, long lTick)
    {
            return createNoteEvent(ShortMessage.NOTE_OFF, pitch, 0, lTick);
    }

    private MidiEvent createNoteEvent(int nCommand, int nKey, int intensity, long lTick)
    {
            ShortMessage message = new ShortMessage();
            try
            {
                    message.setMessage(nCommand, 0,          // always on channel 1
```

```
                                                    nKey,
                                                    intensity);
            }
            catch (InvalidMidiDataException e)
            {
                    e.printStackTrace();
                    System.exit(1);
            }
            MidiEvent           event = new MidiEvent(message,
                                                                                    lTick);
            return event;
    }

    private long getDuration(double duration)
    {
            long d = Math.round((duration/beatSig) * beatTicks);
            return d;
    }

    private List<MidiEvent> addNote(int startTick, int pitch, long lTick, int intensity)
    {
            List<MidiEvent> list = new LinkedList<MidiEvent>();
            System.out.println(startTick);
            System.out.println(startTick + lTick);

            list.add(createNoteOnEvent( pitch, intensity, startTick));
```

```
                list.add(createNoteOffEvent(pitch, startTick + lTick));



                return list;

        }



        private void makeTrack()

        {

                int tick;

                Track t;


t = createTrack(0);
tick = 0;

t.add(createNoteOnEvent(82, tick,100));
tick += getDuration(0.826013580507);
t.add(createNoteOffEvent(82, tick));

t.add(createNoteOnEvent(73, tick,100));
tick += getDuration(0.410471260708);
t.add(createNoteOffEvent(73, tick));

t.add(createNoteOnEvent(83, tick,100));
tick += getDuration(0.274941594302);
t.add(createNoteOffEvent(83, tick));

t.add(createNoteOnEvent(84, tick,100));
tick += getDuration(0.152787091538);
t.add(createNoteOffEvent(84, tick));

t.add(createNoteOnEvent(73, tick,100));
tick += getDuration(0.70773552045);
t.add(createNoteOffEvent(73, tick));

t.add(createNoteOnEvent(73, tick,100));
tick += getDuration(0.658851879608);
t.add(createNoteOffEvent(73, tick));

t.add(createNoteOnEvent(73, tick,100));
tick += getDuration(0.387304924567);
t.add(createNoteOffEvent(73, tick));

t.add(createNoteOnEvent(53, tick,100));
tick += getDuration(0.292969708311);
```

```
t.add(createNoteOffEvent(53, tick));

t.add(createNoteOnEvent(59, tick,100));
tick += getDuration(0.936334469466);
t.add(createNoteOffEvent(59, tick));

t.add(createNoteOnEvent(73, tick,100));
tick += getDuration(0.538295195376);
t.add(createNoteOffEvent(73, tick));

t.add(createNoteOnEvent(65, tick,100));
tick += getDuration(0.300930844192);
t.add(createNoteOffEvent(65, tick));

t.add(createNoteOnEvent(76, tick,100));
tick += getDuration(0.461248747074);
t.add(createNoteOffEvent(76, tick));

t.add(createNoteOnEvent(61, tick,100));
tick += getDuration(0.745820055339);
t.add(createNoteOffEvent(61, tick));

t.add(createNoteOnEvent(85, tick,100));
tick += getDuration(0.529343973776);
t.add(createNoteOffEvent(85, tick));

t.add(createNoteOnEvent(50, tick,100));
tick += getDuration(0.580999267841);
t.add(createNoteOffEvent(50, tick));

t.add(createNoteOnEvent(52, tick,100));
tick += getDuration(0.660608927158);
t.add(createNoteOffEvent(52, tick));

t.add(createNoteOnEvent(64, tick,100));
tick += getDuration(0.207331203189);
t.add(createNoteOffEvent(64, tick));

t.add(createNoteOnEvent(54, tick,100));
tick += getDuration(0.565841889782);
t.add(createNoteOffEvent(54, tick));

t.add(createNoteOnEvent(57, tick,100));
tick += getDuration(0.515829856547);
t.add(createNoteOffEvent(57, tick));

t.add(createNoteOnEvent(78, tick,100));
tick += getDuration(0.866932376666);
t.add(createNoteOffEvent(78, tick));

t.add(createNoteOnEvent(79, tick,100));
tick += getDuration(0.5592264959);
```

```
t.add(createNoteOffEvent(79, tick));

t.add(createNoteOnEvent(70, tick,100));
tick += getDuration(0.180279129322);
t.add(createNoteOffEvent(70, tick));

t.add(createNoteOnEvent(57, tick,100));
tick += getDuration(0.755500327029);
t.add(createNoteOffEvent(57, tick));

t.add(createNoteOnEvent(64, tick,100));
tick += getDuration(0.453569625573);
t.add(createNoteOffEvent(64, tick));

t.add(createNoteOnEvent(52, tick,100));
tick += getDuration(0.280285758083);
t.add(createNoteOffEvent(52, tick));

t.add(createNoteOnEvent(70, tick,100));
tick += getDuration(0.0728667338137);
t.add(createNoteOffEvent(70, tick));

t.add(createNoteOnEvent(55, tick,100));
tick += getDuration(0.00721466811896);
t.add(createNoteOffEvent(55, tick));

t.add(createNoteOnEvent(72, tick,100));
tick += getDuration(0.141909206056);
t.add(createNoteOffEvent(72, tick));

t.add(createNoteOnEvent(80, tick,100));
tick += getDuration(0.0479922013964);
t.add(createNoteOffEvent(80, tick));

t.add(createNoteOnEvent(67, tick,100));
tick += getDuration(0.135805339098);
t.add(createNoteOffEvent(67, tick));

t.add(createNoteOnEvent(63, tick,100));
tick += getDuration(0.175187103795);
t.add(createNoteOffEvent(63, tick));

t.add(createNoteOnEvent(81, tick,100));
tick += getDuration(0.134299212333);
t.add(createNoteOffEvent(81, tick));

t.add(createNoteOnEvent(85, tick,100));
tick += getDuration(0.164700413701);
t.add(createNoteOffEvent(85, tick));

t.add(createNoteOnEvent(74, tick,100));
tick += getDuration(0.139096893038);
```

```
t.add(createNoteOffEvent(74, tick));

t.add(createNoteOnEvent(59, tick,100));
tick += getDuration(0.167372049593);
t.add(createNoteOffEvent(59, tick));

t.add(createNoteOnEvent(59, tick,100));
tick += getDuration(0.154063477118);
t.add(createNoteOffEvent(59, tick));

t.add(createNoteOnEvent(67, tick,100));
tick += getDuration(0.0370862554298);
t.add(createNoteOffEvent(67, tick));

t.add(createNoteOnEvent(60, tick,100));
tick += getDuration(0.00691426039141);
t.add(createNoteOffEvent(60, tick));

t.add(createNoteOnEvent(63, tick,100));
tick += getDuration(0.0072109969814);
t.add(createNoteOffEvent(63, tick));

t.add(createNoteOnEvent(76, tick,100));
tick += getDuration(0.166569371073);
t.add(createNoteOffEvent(76, tick));

t.add(createNoteOnEvent(49, tick,100));
tick += getDuration(0.0444297650141);
t.add(createNoteOffEvent(49, tick));

t.add(createNoteOnEvent(87, tick,100));
tick += getDuration(0.609505276845);
t.add(createNoteOffEvent(87, tick));

t.add(createNoteOnEvent(78, tick,100));
tick += getDuration(0.098186875363);
t.add(createNoteOffEvent(78, tick));

t.add(createNoteOnEvent(62, tick,100));
tick += getDuration(0.366415052502);
t.add(createNoteOffEvent(62, tick));

t.add(createNoteOnEvent(86, tick,100));
tick += getDuration(0.347853356232);
t.add(createNoteOffEvent(86, tick));

t.add(createNoteOnEvent(59, tick,100));
tick += getDuration(0.772057033738);
t.add(createNoteOffEvent(59, tick));

t.add(createNoteOnEvent(70, tick,100));
tick += getDuration(0.750997457434);
```

```
t.add(createNoteOffEvent(70, tick));

t.add(createNoteOnEvent(50, tick,100));
tick += getDuration(0.648743370123);
t.add(createNoteOffEvent(50, tick));

t.add(createNoteOnEvent(66, tick,100));
tick += getDuration(0.482290749398);
t.add(createNoteOffEvent(66, tick));

t.add(createNoteOnEvent(82, tick,100));
tick += getDuration(0.479565446144);
t.add(createNoteOffEvent(82, tick));

t.add(createNoteOnEvent(53, tick,100));
tick += getDuration(0.612900008557);
t.add(createNoteOffEvent(53, tick));

t.add(createNoteOnEvent(78, tick,100));
tick += getDuration(0.791643988338);
t.add(createNoteOffEvent(78, tick));

t.add(createNoteOnEvent(74, tick,100));
tick += getDuration(0.637736555426);
t.add(createNoteOffEvent(74, tick));

t.add(createNoteOnEvent(66, tick,100));
tick += getDuration(0.256963565229);
t.add(createNoteOffEvent(66, tick));

t.add(createNoteOnEvent(61, tick,100));
tick += getDuration(0.5504551374);
t.add(createNoteOffEvent(61, tick));

t.add(createNoteOnEvent(76, tick,100));
tick += getDuration(0.404637594962);
t.add(createNoteOffEvent(76, tick));

t.add(createNoteOnEvent(81, tick,100));
tick += getDuration(0.401836026569);
t.add(createNoteOffEvent(81, tick));

t.add(createNoteOnEvent(62, tick,100));
tick += getDuration(0.265903638555);
t.add(createNoteOffEvent(62, tick));

t.add(createNoteOnEvent(80, tick,100));
tick += getDuration(0.0762503457648);
t.add(createNoteOffEvent(80, tick));

t.add(createNoteOnEvent(73, tick,100));
tick += getDuration(0.859929996582);
```

```
t.add(createNoteOffEvent(73, tick));

t = createTrack(0);
tick = 0;

t.add(createNoteOnEvent(80, tick,100));
tick += getDuration(0.348403386846);
t.add(createNoteOffEvent(80, tick));

t.add(createNoteOnEvent(64, tick,100));
tick += getDuration(0.242046537098);
t.add(createNoteOffEvent(64, tick));

t.add(createNoteOnEvent(66, tick,100));
tick += getDuration(0.33017348879);
t.add(createNoteOffEvent(66, tick));

t.add(createNoteOnEvent(86, tick,100));
tick += getDuration(0.398045042949);
t.add(createNoteOffEvent(86, tick));

t.add(createNoteOnEvent(91, tick,100));
tick += getDuration(0.274469515844);
t.add(createNoteOffEvent(91, tick));

t.add(createNoteOnEvent(72, tick,100));
tick += getDuration(0.56778241236);
t.add(createNoteOffEvent(72, tick));

t.add(createNoteOnEvent(85, tick,100));
tick += getDuration(0.467318431648);
t.add(createNoteOffEvent(85, tick));

t.add(createNoteOnEvent(88, tick,100));
tick += getDuration(0.677288377887);
t.add(createNoteOffEvent(88, tick));

t.add(createNoteOnEvent(68, tick,100));
tick += getDuration(0.44013444848);
t.add(createNoteOffEvent(68, tick));

t.add(createNoteOnEvent(62, tick,100));
tick += getDuration(0.340686000789);
t.add(createNoteOffEvent(62, tick));

t.add(createNoteOnEvent(63, tick,100));
tick += getDuration(0.10645771829);
t.add(createNoteOffEvent(63, tick));

t.add(createNoteOnEvent(74, tick,100));
tick += getDuration(0.331738122927);
t.add(createNoteOffEvent(74, tick));
```

```
t.add(createNoteOnEvent(86, tick,100));
tick += getDuration(0.0367384587954);
t.add(createNoteOffEvent(86, tick));

t.add(createNoteOnEvent(67, tick,100));
tick += getDuration(0.577930283321);
t.add(createNoteOffEvent(67, tick));

t.add(createNoteOnEvent(75, tick,100));
tick += getDuration(0.706498994756);
t.add(createNoteOffEvent(75, tick));

t.add(createNoteOnEvent(79, tick,100));
tick += getDuration(0.444509031737);
t.add(createNoteOffEvent(79, tick));

t.add(createNoteOnEvent(88, tick,100));
tick += getDuration(0.560010774878);
t.add(createNoteOffEvent(88, tick));

t.add(createNoteOnEvent(64, tick,100));
tick += getDuration(0.0225457597379);
t.add(createNoteOffEvent(64, tick));

t.add(createNoteOnEvent(78, tick,100));
tick += getDuration(0.423586837052);
t.add(createNoteOffEvent(78, tick));

t.add(createNoteOnEvent(77, tick,100));
tick += getDuration(0.401446383662);
t.add(createNoteOffEvent(77, tick));

t.add(createNoteOnEvent(65, tick,100));
tick += getDuration(0.0149948228038);
t.add(createNoteOffEvent(65, tick));

t.add(createNoteOnEvent(78, tick,100));
tick += getDuration(0.421701409369);
t.add(createNoteOffEvent(78, tick));

t.add(createNoteOnEvent(84, tick,100));
tick += getDuration(0.137603959644);
t.add(createNoteOffEvent(84, tick));

t.add(createNoteOnEvent(72, tick,100));
tick += getDuration(0.146720167214);
t.add(createNoteOffEvent(72, tick));

t.add(createNoteOnEvent(85, tick,100));
tick += getDuration(0.0901880537044);
t.add(createNoteOffEvent(85, tick));
```

```
t.add(createNoteOnEvent(90, tick,100));
tick += getDuration(0.113343573074);
t.add(createNoteOffEvent(90, tick));

t.add(createNoteOnEvent(77, tick,100));
tick += getDuration(0.0166847537876);
t.add(createNoteOffEvent(77, tick));

t.add(createNoteOnEvent(77, tick,100));
tick += getDuration(0.0824990405486);
t.add(createNoteOffEvent(77, tick));

t.add(createNoteOnEvent(84, tick,100));
tick += getDuration(0.0915373665842);
t.add(createNoteOffEvent(84, tick));

t.add(createNoteOnEvent(77, tick,100));
tick += getDuration(0.11929377251);
t.add(createNoteOffEvent(77, tick));

t.add(createNoteOnEvent(80, tick,100));
tick += getDuration(0.0719583877745);
t.add(createNoteOffEvent(80, tick));

t.add(createNoteOnEvent(85, tick,100));
tick += getDuration(0.00123831336157);
t.add(createNoteOffEvent(85, tick));

t.add(createNoteOnEvent(85, tick,100));
tick += getDuration(0.00769014966663);
t.add(createNoteOffEvent(85, tick));

t.add(createNoteOnEvent(80, tick,100));
tick += getDuration(0.0217089493628);
t.add(createNoteOffEvent(80, tick));

t.add(createNoteOnEvent(89, tick,100));
tick += getDuration(0.0461267634802);
t.add(createNoteOffEvent(89, tick));

t.add(createNoteOnEvent(70, tick,100));
tick += getDuration(0.0112398891642);
t.add(createNoteOffEvent(70, tick));

t.add(createNoteOnEvent(71, tick,100));
tick += getDuration(0.0911034412002);
t.add(createNoteOffEvent(71, tick));

t.add(createNoteOnEvent(86, tick,100));
tick += getDuration(0.1106904732);
t.add(createNoteOffEvent(86, tick));
```

```
t.add(createNoteOnEvent(62, tick,100));
tick += getDuration(0.108846594624);
t.add(createNoteOffEvent(62, tick));

t.add(createNoteOnEvent(69, tick,100));
tick += getDuration(0.0605796469785);
t.add(createNoteOffEvent(69, tick));

t.add(createNoteOnEvent(89, tick,100));
tick += getDuration(0.109216410655);
t.add(createNoteOffEvent(89, tick));

t.add(createNoteOnEvent(83, tick,100));
tick += getDuration(0.0321472265673);
t.add(createNoteOffEvent(83, tick));

t.add(createNoteOnEvent(84, tick,100));
tick += getDuration(0.370787238675);
t.add(createNoteOffEvent(84, tick));

t.add(createNoteOnEvent(84, tick,100));
tick += getDuration(0.424424409783);
t.add(createNoteOffEvent(84, tick));

t.add(createNoteOnEvent(83, tick,100));
tick += getDuration(0.137116250647);
t.add(createNoteOffEvent(83, tick));

t.add(createNoteOnEvent(66, tick,100));
tick += getDuration(0.66746747511);
t.add(createNoteOffEvent(66, tick));

t.add(createNoteOnEvent(63, tick,100));
tick += getDuration(0.707105521098);
t.add(createNoteOffEvent(63, tick));

t.add(createNoteOnEvent(70, tick,100));
tick += getDuration(0.28375808157);
t.add(createNoteOffEvent(70, tick));

t.add(createNoteOnEvent(70, tick,100));
tick += getDuration(0.0499682528146);
t.add(createNoteOffEvent(70, tick));

t.add(createNoteOnEvent(72, tick,100));
tick += getDuration(0.668592366658);
t.add(createNoteOffEvent(72, tick));

t.add(createNoteOnEvent(77, tick,100));
tick += getDuration(0.0598447326399);
t.add(createNoteOffEvent(77, tick));
```

```
t.add(createNoteOnEvent(77, tick,100));
tick += getDuration(0.0969141388041);
t.add(createNoteOffEvent(77, tick));

t.add(createNoteOnEvent(74, tick,100));
tick += getDuration(0.741099810896);
t.add(createNoteOffEvent(74, tick));

t.add(createNoteOnEvent(78, tick,100));
tick += getDuration(0.486310232315);
t.add(createNoteOffEvent(78, tick));

t.add(createNoteOnEvent(86, tick,100));
tick += getDuration(0.726460112579);
t.add(createNoteOffEvent(86, tick));

t.add(createNoteOnEvent(63, tick,100));
tick += getDuration(0.484524459715);
t.add(createNoteOffEvent(63, tick));

t.add(createNoteOnEvent(72, tick,100));
tick += getDuration(0.450624486347);
t.add(createNoteOffEvent(72, tick));

t.add(createNoteOnEvent(81, tick,100));
tick += getDuration(0.434749231484);
t.add(createNoteOffEvent(81, tick));

t.add(createNoteOnEvent(75, tick,100));
tick += getDuration(0.343787383822);
t.add(createNoteOffEvent(75, tick));

t.add(createNoteOnEvent(65, tick,100));
tick += getDuration(0.062066755002);
t.add(createNoteOffEvent(65, tick));
t.add(createNoteOnEvent(0, tick, 0));
tick += getDuration(0.5);
t.add(createNoteOffEvent(0, tick));

        }
public static void main(String[] args)

        {

new Output(90, 0.25);

        }

}
```

Appendix E: Complete Code Listing

E.1 m.ml

Author: Jiaying Xu

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

E.2 scanner.mll

Author: Monica Ramirez-Santana

```
{ open Parser } (* Get the token types *)

rule token = parse
            [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
            | "/*" { comment lexbuf } (* Comments *)
            | "//" { singlecomment lexbuf }
            | '(' { LPAREN }
            | ')' { RPAREN } (* punctuation *)
            | '{' { LBRACE }
            | '}' { RBRACE }
            | ';' { SEMI }
            | ',' { COMMA }
            | '.' { DOT }
            | '+' { PLUS } (* started here *)
            | '-' { MINUS }
            | '*' { TIMES }
            | '/' { DIVIDE }
            | '%' { MOD }
            | "+=" { PLUSEQ }
            | "-=" { MINUSEQ }
            | "*=" { TIMESEQ }
            | "/=" { DIVIDEEQ }
            | "%=" { MODEQ }
            | '=' { ASSIGN }
            | '!' { NOT }
            | "++" { PLUSPLUS }
            | "--" { MINUSMINUS }
            | "==" { EQ }
            | "!=" { NEQ }
            | '<' { LT }
            | "<=" { LEQ }
            | ">" { GT }
            | ">=" { GEQ }
            | "&&" { AND }
```

```
| "||" { OR }
| "if" { IF } (* keywords *)
| "else" { ELSE }
| "for" { FOR }
| "while" { WHILE }
| "return" { RETURN }
| "void" { DATATYPE("void") }
| "int" { DATATYPE("int") }
| "float" { DATATYPE("float") }
| "bool" { DATATYPE("bool") }
| "note" { DATATYPE("note") }
| "chord" { DATATYPE("chord") }
| "staff" { DATATYPE("staff") }
| "part" { DATATYPE("part") }
| "true"|"false" as boollit { BOOLLITERAL(bool_of_string boollit) }
| (['a'-'g' 'A'-'G']['s' 'f' 'S' 'F']?['0'-'9'])|('r'|'R') as pitchlit { PITCHLITERAL(pitchlit) }
| eof { EOF } (* Endoffile *)
| ['0'-'9']+ as lxm { INTLITERAL(int_of_string lxm) } (* integers *)
| ((['0'-'9']+'.'['0'-'9']*)) as floatlit { FLOATLITERAL(float_of_string floatlit) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| _ as char { raise (Failure("illegal character: " ^ Char.escaped char)) }

and comment = parse
"*/" { token lexbuf } (* Endofcomment *)
| _ { comment lexbuf } (* Eat everything else *)

and singlecomment = parse
['\r' '\n'] {token lexbuf } (* Endofcomment *)
| _ {singlecomment lexbuf } (* Eat everything else *)
```

E.3 parser.mly

Author: Monica Ramirez-Santana, Yiling Hu

```
%{ open Ast %}

%token LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET
%token SEMI COMMA DOT
%token PLUS MINUS TIMES DIVIDE MOD PLUSPLUS MINUSMINUS
%token PLUSEQ MINUSEQ TIMESEQ DIVIDEEQ MODEQ
%token EQ NEQ LT LEQ GT GEQ AND NOT OR ASSIGN
%token IF ELSE ELSEIF FOR WHILE RETURN
%token INT VOID FLOAT BOOL NOTE CHORD STAFF PART
%token <int> INTLITERAL
%token <float> FLOATLITERAL
%token <bool> BOOLLITERAL
%token <string> ID
%token <string> DATATYPE
%token <string> PITCHLITERAL
```

```
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%nonassoc LPAREN
%left PLUSEQ MINUSEQ
%left TIMESEQ DIVIDEEQ MODEQ
%right ASSIGN
%left OR
%left AND
%left EQ NEQ
%left LT GT LEQ GEQ
%right NOT
%left PLUS MINUS
%left TIMES DIVIDE MOD
%left PLUSPLUS MINUSMINUS
%start program
%type <Ast.program> program

%%

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

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

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

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

vdecl_list:
        /* nothing */    { [] }
        | vdecl_list vdecl { $2 :: $1 }

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

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

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

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

expr_opt:
        /* nothing */ { NoExpr }
        | expr { $1 }

expr:
        ID { Id($1) }                                                   //x
        | ID DOT ID { MemberAccess($1, $3) }                            //note.pitch
        | INTLITERAL { IntLiteral($1) }                                 //1
        | FLOATLITERAL { FloatLiteral($1) }                             //1.0
        | BOOLLITERAL { BoolLiteral($1) }                               //true
        | PITCHLITERAL { PitchLiteral ($1) }                            //As7
        | expr ASSIGN expr { Assign($1,$3) }                            //x = y
        | expr PLUSEQ expr { Assign($1, BinOp($1, Add, $3)) }           //x += y
        | expr MINUSEQ expr { Assign($1, BinOp($1, Sub, $3)) }          //x -= y
        | expr TIMESEQ expr { Assign($1, BinOp($1, Mult, $3)) }         //x *= y
        | expr DIVIDEEQ expr { Assign($1, BinOp($1, Div, $3)) }         //x /=y
        | expr MODEQ expr { Assign($1, BinOp($1, Mod, $3)) }            //x %= y
        | expr PLUS expr { BinOp($1, Add, $3) }                         //x + y
        | expr MINUS expr { BinOp($1, Sub, $3) }                        //x - y
        | expr TIMES expr { BinOp($1, Mult, $3) }                       //x * y
        | expr DIVIDE expr { BinOp($1, Div, $3) }                       //x / y
        | expr MOD expr { BinOp($1, Mod, $3) }                          //x % y
        | expr AND expr { BinOp($1, And, $3) }                          //x && y
        | expr OR expr { BinOp($1, Or, $3) }                            //x || y
        | expr EQ expr { BinOp($1, Eq, $3) }                            //x == y
        | expr NEQ expr { BinOp($1, NEq, $3) }                          //x != y
        | expr LT expr { BinOp($1, Less, $3) }                          //x < y
        | expr LEQ expr { BinOp($1, LEq, $3) }                          //x <= y
        | expr GT expr { BinOp($1, Greater, $3) }                       //x > y
```

```
        | expr GEQ expr { BinOp($1, GEq, $3) }                                    //x >= y
        | NOT expr { Not($2) }                                                   //!x
        | expr PLUSPLUS { Assign($1, BinOp($1, Add, IntLiteral(1))) }            //x++
        | expr MINUSMINUS { Assign($1, BinOp($1, Sub, IntLiteral(1))) }          //x--
        | LPAREN expr RPAREN { $2 }     //(x), expression in parenthesis is the same as the expression
        | ID LPAREN actuals_opt RPAREN { Call($1, $3) }                          //x(...), function call

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

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

E.4 ast.ml

Author: Monica Ramirez-Santana, Yiling Hu

```
type op =
        Add | Sub | Mult | Div | Mod
        | And | Or | Eq | NEq | Less | LEq | Greater | GEq

type expr = (* Expressions *)
        Id of string (* foo *)
        | MemberAccess of string * string (* foo.intensity *)
        | IntLiteral of int (* 42 *)
        | FloatLiteral of float (* 42.0 *)
        | BoolLiteral of bool (* true *)
        | PitchLiteral of string (* As7 *)
        | Assign of expr * expr (* x = y *)
        | BinOp of expr * op * expr (* x + y *)
        | Not of expr (* !x *)
        | Call of string * expr list (* foo(x, y) *)
        | NoExpr (* for (;;) *)

type stmt = (* Statements *)
        Expr of expr (* foo = bar + 3; *)
        | Return of expr (* return 42; *)
        | Block of stmt list (* { ... } *)
        | If of expr * stmt * stmt (* if (foo == 42) {} else {} *)
        | For of expr * expr * expr * stmt (* for (i=0;i<10;i=i+1) { ... } *)
        | While of expr * stmt (* while (i<10) { i = i + 1 } *)

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

```
type var_decl = {
        varname : string; (* Name of the variable *)
        vartype : string; (* Name of variable type *)
}

type func_decl = {
        fname : string; (* Name of the function *)
        rettype : string; (* Name of return type *)
        formals : par_decl list; (* Formal argument names *)
        locals : var_decl list; (* Locally defined variables *)
        body : stmt list;
}

type program = var_decl list * func_decl list (* global vars, funcs *)
```

E.5 interpret.ml

Authors: Jiaying Xu, Yiling Hu

```
open Ast
open Printf

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


type note = {
        mutable pitch : int;
        mutable intensity : int;
        mutable duration : float;
}

type chord = {
        mutable notelist : note list;
}

type staff = {
        mutable instrument : int;
        mutable chordlist : chord list;
}

type part = {
        mutable bpm : int;
        mutable beatsig : float;
        mutable stafflist : staff list;
}
```

```
type mtype = Int of int | Float of float | Bool of bool | Note of note | Chord of chord | Staff of staff |
Part of part

exception ReturnException of mtype * mtype NameMap.t
(* check pitch between *)
let getType v =
	match v with
		Int(v) -> "int"
		| Float(v) -> "float"
		| Bool(v) -> "bool"
		| Note(v) -> "note"
		| Chord(v) -> "chord"
		| Staff(v) -> "staff"
		| Part(v) -> "part"
(* fix names *)
let getInt v =
	match v with
		Int(v) -> v
		| _ -> 0

let getFloat v =
	match v with
		Float(v) -> v
		| _ -> 0.0

let getBool v =
	match v with
		Bool(v) -> v
		| _ -> false

let getNote v =
	match v with
		Note(v) -> v
		| _ -> {pitch=128; intensity=0; duration=0.0}

let getChord v =
	match v with
		Chord(v) -> v
		| _ -> {notelist=[]}

let getStaff v =
	match v with
		Staff(v) -> v
		| _ -> {instrument=0; chordlist=[]}

let getPart v =
	match v with
		Part(v) -> v
```

```
                    | _ -> {bpm=0; beatsig=0.0; stafflist=[]}

let initIdentifier t =
  match t with
    "int" -> Int(0)
    | "float" -> Float(0.0)
    | "bool" -> Bool(false)
    | "note" -> Note({pitch=128; intensity=0; duration=0.0})
    | "chord" -> Chord({notelist=[]})
    | "staff" -> Staff({instrument=0; chordlist=[]})
    | "part" -> Part({bpm=0; beatsig=0.0; stafflist=[]})
    | _ -> Bool(false)

let import_decl =
"
import java.io.File;
import java.io.IOException;
import java.math.BigInteger;
import java.util.LinkedList;
import java.util.List;
import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.MetaMessage;
import javax.sound.midi.MidiEvent;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.Sequence;
import javax.sound.midi.ShortMessage;
import javax.sound.midi.Track;
"

let class_start =
"
public class Output
{
        Sequence sequence = null;
        List<Track> trackList;
        int tempo;
        double beatSig;
        int beatTicks;

        public Output(int bpm, double beatsig)
        {
                this.tempo = bpm;
                this.beatSig = beatsig;
                beatTicks = (int) (beatsig/(1.0/32.0));
                trackList = new LinkedList<Track>();

                try
                {
```

```
                sequence = new Sequence(Sequence.PPQ, beatTicks);
        }
        catch (InvalidMidiDataException e)
        {
                e.printStackTrace();
                System.exit(1);
        }

        makeTrack();

        try
        {
                MidiSystem.write(sequence, 1, new File(\"output.mid\"));
                System.out.println(\"file written\");
        }
        catch (IOException e)
        {
                e.printStackTrace();
                System.exit(1);
        }
}

private byte[] getTempo(int bpm)
{
        int ms = 60000000;
        Integer i = new Integer(ms/bpm);
        BigInteger bi = new BigInteger(i.toString());
        return bi.toByteArray();
}

private Track createTrack(int instrument)
{
        Track t = sequence.createTrack();
        try
        {
                MetaMessage mt = new MetaMessage();
                byte[] bt = getTempo(tempo);
                mt.setMessage(0x51 ,bt, bt.length);
                MidiEvent me = new MidiEvent(mt, 0);
                t.add(me);

                ShortMessage sm = new ShortMessage();
                sm.setMessage(192, instrument, 0);
                me = new MidiEvent(sm, 0);
                t.add(me);
        } catch (Exception e)
        {
                e.printStackTrace();
```

```
		}
		return t;
}

private MidiEvent createNoteOnEvent(int pitch, long lTick, int intensity)
{
		return createNoteEvent(ShortMessage.NOTE_ON, pitch, intensity, lTick);
}

private MidiEvent createNoteOffEvent(int pitch, long lTick)
{
		return createNoteEvent(ShortMessage.NOTE_OFF, pitch, 0, lTick);
}

private MidiEvent createNoteEvent(int nCommand, int nKey, int intensity, long lTick)
{
		ShortMessage message = new ShortMessage();
		try
		{
			message.setMessage(nCommand, 0,	// always on channel 1
					nKey,
					intensity);
		}
		catch (InvalidMidiDataException e)
		{
			e.printStackTrace();
			System.exit(1);
		}
		MidiEvent	event = new MidiEvent(message,
								lTick);
		return event;
}

private long getDuration(double duration)
{
		long d = Math.round((duration/beatSig) * beatTicks);
		return d;
}

private List<MidiEvent> addNote(int startTick, int pitch, long lTick, int intensity)
{
		List<MidiEvent> list = new LinkedList<MidiEvent>();
		System.out.println(startTick);
		System.out.println(startTick + lTick);

		list.add(createNoteOnEvent( pitch, intensity, startTick));
		list.add(createNoteOffEvent(pitch, startTick + lTick));
```

```
		return list;
	}
"

let staff_start =
"
	private void makeTrack()
	{
		int tick;
		Track t;
"

let staff_end =
"
	}
"

let main_start =
"
public static void main(String[] args)
	{
"

let main_end =
"
	}
"

let class_end =
"
}
"

(* Main entry point: run a program *)
let run (vars, funcs) =
	(* Put function declarations in a symbol table *)
	let func_decls = List.fold_left
		(fun funcs fdecl -> NameMap.add fdecl.fname fdecl funcs)
		NameMap.empty funcs
	in

	(* Invoke a function and return an updated global symbol table *)
	let rec call fdecl actuals globals =

	(* Evaluate an expression and return (value, updated environment) *)
	let rec eval env = function
		IntLiteral(i) -> Int i, env
		| FloatLiteral(i) -> Float i, env
```

```
            | BoolLiteral(i) -> Bool i, env
            | PitchLiteral(i) -> (
            if String.length i == 3 || String.length i == 2 then
    (let octave =
      int_of_string (String.sub i (String.length i-1) 1)
     in
    (let sfvalue =
            if String.length i == 3 then
        (if String.sub (String.lowercase i) 1 1 = "s" then 1
        else if String.sub (String.lowercase i) 1 1 = "f" then -1
        else raise (Failure ("invalid pitch: " ^ i)))
      else 0
     in
    (match String.sub (String.lowercase i) 0 1 with
    "a" -> Int ((9 + sfvalue) + (12 * (octave + 1)))
    | "b" -> Int ((11 + sfvalue) + (12 * (octave + 1)))
    | "c" -> Int ((0 + sfvalue) + (12 * (octave + 1)))
    | "d" -> Int ((2 + sfvalue) + (12 * (octave + 1)))
    | "e" -> Int ((4 + sfvalue) + (12 * (octave + 1)))
    | "f" -> Int ((5 + sfvalue) + (12 * (octave + 1)))
    | "g" -> Int ((7 + sfvalue) + (12 * (octave + 1)))
    | _ -> raise (Failure ("invalid pitch: " ^ i))
    )))
   else if (String.lowercase i) = "r" then Int 128
   else raise (Failure ("invalid pitch: " ^ i))), env
| NoExpr ->Bool true, env (* must be nonzero for the for loop predicate *) (* is this correct *)
            | Id(var) ->
                    let locals, globals = env in
                    if NameMap.mem var locals then
                            (NameMap.find var locals), env
                    else if NameMap.mem var globals then
                            (NameMap.find var globals), env
                    else raise (Failure ("undeclared identifier: " ^ var))
            | MemberAccess(var, i) ->
                    let v, env = eval env (Id var) in
                            let vType = getType v in
                            (match vType with
                              | "note" ->
                                      (match i with
                                        "pitch" -> Int (getNote v).pitch
                                        | "intensity" -> Int (getNote v).intensity
                                        | "duration" -> Float (getNote v).duration
                                        | _ -> raise (Failure ("invalid property of note: " ^ i)))
                              | "staff" ->
                                      (match i with
                                              "instrument" -> Int (getStaff v).instrument
                                        | _ -> raise (Failure ("invalid property of staff: " ^ i)))
                              | "part" ->
```

```
                        (match i with
                                "bpm" -> Int (getPart v).bpm
                          | "beatsig" -> Float (getPart v).beatsig
                          | _ -> raise (Failure ("invalid property of part: " ^ i)))
                  | _ -> raise (Failure ("cannot access " ^ var ^ "." ^ i))), env
| BinOp(e1, op, e2) ->
        let v1, env = eval env e1 in
        let v2, env = eval env e2 in
let v1Type = getType v1 in
let v2Type = getType v2 in
if v1Type = v2Type then
        (match op with
                Add ->
                if v1Type = "int" then
                        Int (getInt v1 + getInt v2)
        else if v1Type = "float" then
                        Float (getFloat v1 +. getFloat v2)
        else raise (Failure ("incorrect type: " ^ v1Type ^ " + " ^ v2Type))
                | Sub ->
                if v1Type = "int" then
                        Int (getInt v1 - getInt v2)
        else if v1Type = "float" then
                        Float (getFloat v1 -. getFloat v2)
        else raise (Failure ("incorrect type: " ^ v1Type ^ " - " ^ v2Type))
                | Mult ->
                if v1Type = "int" then
                        Int (getInt v1 * getInt v2)
        else if v1Type = "float" then
                        Float (getFloat v1 *. getFloat v2)
        else raise (Failure ("incorrect type: " ^ v1Type ^ " * " ^ v2Type))
                | Div ->
                if v1Type = "int" then
                        Int (getInt v1 / getInt v2)
        else if v1Type = "float" then
                        Float (getFloat v1 /. getFloat v2)
        else raise (Failure ("incorrect type: " ^ v1Type ^ " / " ^ v2Type))
                | Mod ->
                if v1Type = "int" then
                        Int (getInt v1 mod getInt v2)
        else raise (Failure ("incorrect type: " ^ v1Type ^ " % " ^ v2Type))
                | And ->
                if v1Type = "bool" then
                        Bool (getBool v1 && getBool v2)
        else raise (Failure ("incorrect type: " ^ v1Type ^ " && " ^ v2Type))
                | Or ->
                if v1Type = "bool" then
                        Bool (getBool v1 || getBool v2)
        else raise (Failure ("incorrect type: " ^ v1Type ^ " || " ^ v2Type))
```

```
                | Eq ->
        if v1Type = "int" then
                        Bool (getInt v1 = getInt v2)
        else if v1Type = "float" then
                        Bool (getFloat v1 = getFloat v2)
        else if v1Type = "bool" then
                        Bool (getBool v1 = getBool v2)
        else if v1Type = "note" then
                        Bool (getNote v1 = getNote v2)
        else if v1Type = "chord" then
                        Bool (getChord v1 = getChord v2)
        else if v1Type = "staff" then
                        Bool (getStaff v1 = getStaff v2)
        else if v1Type = "part" then
                        Bool (getPart v1 = getPart v2)
        else raise (Failure ("incorrect type: " ^ v1Type ^ " == " ^ v2Type))

        | NEq ->
        if v1Type = "int" then
                        Bool (getInt v1 != getInt v2)
        else if v1Type = "float" then
                        Bool (getFloat v1 != getFloat v2)
        else if v1Type = "bool" then
                        Bool (getBool v1 != getBool v2)
        else if v1Type = "note" then
                        Bool (getNote v1 != getNote v2)
        else if v1Type = "chord" then
                        Bool (getChord v1 != getChord v2)
        else if v1Type = "staff" then
                        Bool (getStaff v1 != getStaff v2)
        else if v1Type = "part" then
                        Bool (getPart v1 != getPart v2)
        else raise (Failure ("incorrect type: " ^ v1Type ^ " != " ^ v2Type))
        | Less ->
        if v1Type = "int" then
                        Bool (getInt v1 < getInt v2)
        else if v1Type = "float" then
                        Bool (getFloat v1 < getFloat v2)
        else raise (Failure ("cannot compare: " ^ v1Type ^ " < " ^ v2Type))
| LEq ->
        if v1Type = "int" then
                        Bool (getInt v1 <= getInt v2)
        else if v1Type = "float" then
                        Bool (getFloat v1 <= getFloat v2)
        else raise (Failure ("cannot compare: " ^ v1Type ^ " <= " ^ v2Type))
| Greater ->
        if v1Type = "int" then
                        Bool (getInt v1 > getInt v2)
```

```
                        else if v1Type = "float" then
                                Bool (getFloat v1 > getFloat v2)
                        else raise (Failure ("cannot compare: " ^ v1Type ^ " > " ^ v2Type))
                | GEq ->
                        if v1Type = "int" then
                                Bool (getInt v1 >= getInt v2)
                        else if v1Type = "float" then
                                Bool (getFloat v1 >= getFloat v2)
                        else raise (Failure ("cannot compare: " ^ v1Type ^ " >= " ^ v2Type))), env
         else raise (Failure ("type mismatch: " ^ v1Type ^ " and " ^ v2Type))
        (* must test thoroughly! *)
        | Assign(var, e) ->
                        let var1, env = eval env var in
                        let e1, (locals, globals) = eval env e in
        let v1Info =
        match var with
                        Id (i) -> ("id", (i, ""))
                        | MemberAccess(i, j) -> ("member", (i, j))
          | _ -> raise (Failure ("left side of assignment must be an identifier or member access")) in
        let v1IdType = fst v1Info in
       let v1Name = snd v1Info in
        let v1Type = (* ("note", "locals") *)
                                (if NameMap.mem (fst v1Name) locals then
                                        (getType (NameMap.find (fst v1Name) locals), "locals")
                                else if NameMap.mem (fst v1Name) globals then
                                        (getType (NameMap.find (fst v1Name) globals), "globals")
                                else raise (Failure ("undeclared identifier: " ^ fst v1Name)))
        in
        let v1RetType = getType var1 in
        let e1Type = getType e1 in
                if v1RetType = e1Type then
                match v1RetType with
                                "int" ->
                                        if v1IdType = "id" then
                                                (if snd v1Type = "locals" then
                                                        e1, (NameMap.add (fst v1Name) e1 locals,
globals)
                                                else if snd v1Type = "globals" then
                                                        e1, (locals, NameMap.add (fst v1Name) e1
globals)
                                                else raise (Failure ("fatal error")))
                                        else if v1IdType = "member" then
                                                if fst v1Type = "note" then
                                                        if snd v1Name = "pitch" then
                                                                if getInt e1 >= 0 && getInt e1 <= 128
then
                                                                        if snd v1Type = "locals" then
```

```
                                                    e1, (((getNote
(NameMap.find (fst v1Name) locals)).pitch <- getInt e1); (locals, globals))
                                                else if snd v1Type = "globals"
then
                                                    e1, (((getNote
(NameMap.find (fst v1Name) globals)).pitch <- getInt e1); (locals, globals))
                                                else raise (Failure ("fatal
error"))
                                            else raise (Failure ("invalid note pitch: "
^ string_of_int (getInt e1) ^ ". pitch must be between 0-128."))
                                        else if snd v1Name = "intensity" then (* min
max checking *)
                                            if getInt e1 >= 0 && getInt e1 <= 127
then
                                                if snd v1Type = "locals" then
                                                    e1, (((getNote
(NameMap.find (fst v1Name) locals)).intensity <- getInt e1); (locals, globals))
                                                else if snd v1Type = "globals"
then
                                                    e1, (((getNote
(NameMap.find (fst v1Name) globals)).intensity <- getInt e1); (locals, globals))
                                                else raise (Failure ("undeclared
identifier: " ^ fst v1Name))
                                            else raise (Failure ("invalid note
intensity: " ^ string_of_int (getInt e1) ^ ". intensity must be between 0-127."))
                                        else raise (Failure ("fatal error"))
                                    else if fst v1Type = "staff" && (snd v1Name =
"instrument") then
                                        if snd v1Name = "instrument" then
                                            if getInt e1 >= 0 && getInt e1 <= 127
then
                                                if snd v1Type = "locals" then
                                                    e1, (((getStaff
(NameMap.find (fst v1Name) locals)).instrument <- getInt e1); (locals, globals))
                                                else if snd v1Type = "globals"
then
                                                    e1, (((getStaff
(NameMap.find (fst v1Name) globals)).instrument <- getInt e1); (locals, globals))
                                                else raise (Failure ("fatal
error"))
                                            else raise (Failure ("invalid staff
instrument: " ^ string_of_int (getInt e1) ^ ". instrument must be between 0-127."))
                                        else raise (Failure ("fatal error"))
                                    else if fst v1Type = "part" then
                                        if snd v1Name = "bpm" then
                                            if getInt e1 >= 0 && getInt e1 <= 240
then
                                                if snd v1Type = "locals" then
```

```
                                                                    e1, (((getPart
(NameMap.find (fst v1Name) locals)).bpm <- getInt e1); (locals, globals))
                                                                else if snd v1Type = "globals"
then
                                                                    e1, (((getPart
(NameMap.find (fst v1Name) globals)).bpm <- getInt e1); (locals, globals))
                                                                else raise (Failure ("fatal
error"))
                                                            else raise (Failure ("invalid part bpm: " ^
string_of_int (getInt e1) ^ ". BPM must be between 0-240."))
                                                        else raise (Failure ("fatal error"))
                                                    else raise (Failure ("cannot assign to: " ^ fst v1Type))
                                                else raise (Failure ("cannot assign to: " ^ fst v1Type))
                                            | "float" ->
                                                if v1IdType = "id" then
                                                    (if snd v1Type = "locals" then
                                                        e1, (NameMap.add (fst v1Name) e1 locals,
globals)
                                                    else if snd v1Type = "globals" then
                                                        e1, (locals, NameMap.add (fst v1Name) e1
globals)
                                                    else raise (Failure ("fatal error")))
                                                else if v1IdType = "member" then
                                                    if fst v1Type = "note" then
                                                        if snd v1Name = "duration" then
                                                            if getFloat e1 >= 0.0 && getFloat e1 <=
1.0 then
                                                                if snd v1Type = "locals" then
                                                                    e1, (((getNote
(NameMap.find (fst v1Name) locals)).duration <- getFloat e1); (locals, globals))
                                                                else if snd v1Type = "globals"
then
                                                                    e1, (((getNote
(NameMap.find (fst v1Name) globals)).duration <- getFloat e1); (locals, globals))
                                                                else raise (Failure ("fatal
error"))
                                                            else raise (Failure ("invalid note
duration: " ^ string_of_float (getFloat e1) ^ ". duration must be between 0.0-1.0."))
                                                        else raise (Failure ("fatal error"))
                                                    else if fst v1Type = "part" then
                                                        if snd v1Name = "beatsig" then
                                                            if getFloat e1 = 1.0 || getFloat e1 = 0.5
|| getFloat e1 = 0.25 || getFloat e1 = 0.125 || getFloat e1 = 0.0625 || getFloat e1 = 0.03125 then
                                                                if snd v1Type = "locals" then
                                                                    e1, (((getPart
(NameMap.find (fst v1Name) locals)).beatsig <- getFloat e1); (locals, globals))
                                                                else if snd v1Type = "globals"
then
```

```
                                                                    e1, (((getPart
(NameMap.find (fst v1Name) globals)).beatsig <- getFloat e1); (locals, globals))
                                                                else raise (Failure ("fatal
error"))
                                                            else raise (Failure ("invalid part beat
signature: " ^ string_of_float (getFloat e1) ^ ". beat signature can only be: 1.0, 0.5, 0.25, 0.125, 0.0625,
0.03125."))
                                                        else raise (Failure ("fatal error"))
                                                    else raise (Failure ("cannot assign to: " ^ fst v1Type))
                                                else raise (Failure ("cannot assign to: " ^ fst v1Type))
                                            | _ -> (* bool, note, chord, staff, part *)
                                                if v1IdType = "id" then
                                                    (if snd v1Type = "locals" then
                                                        e1, (NameMap.add (fst v1Name) e1 locals,
globals)
                                                    else if snd v1Type = "globals" then
                                                        e1, (locals, NameMap.add (fst v1Name) e1
globals)
                                                    else raise (Failure ("fatal error")))
                                                else raise (Failure ("cannot assign to: " ^ fst v1Type))
                        else if v1IdType = "id" then
                                raise (Failure ("cannot assign: " ^ fst v1Type ^ " = " ^ e1Type))
                    else if v1IdType = "member" then
                                raise (Failure ("cannot assign: " ^ v1RetType ^ " = " ^ e1Type))
                     else raise (Failure ("fatal error"))
                | Not(var) ->
                        let vnew, envnew = eval env var in
                        if getType vnew = "bool" then
                                        Bool(not (getBool vnew)), envnew
                  else raise (Failure ("type mismatch: !" ^ getType vnew))
                | Call("randint", [e]) ->
                                let v, env = eval env e in
                        if getType v = "int" then
                                                Int(Random.int (getInt v)), env
                    else raise (Failure ("argument of randint must be an integer"))
                        | Call("randfloat", [e]) ->
                                let v, env = eval env e in
                                if getType v = "float" then
                                                Float(Random.float (getFloat v)), env
                        else raise (Failure ("argument of randfloat must be a float"))
                        | Call("add", [var; e]) ->
                        let v1name = match var with
                        Id(i) -> i
                                | _ ->  raise (Failure ("arguments of add must be identifiers")) in
                                ignore (match e with
                        Id(i) -> "id"
                                | _ ->  raise (Failure ("arguments of add must be identifiers")));
                                let var1, env = eval env var in
```

```
                        let e1, (locals, globals) = eval env e in
        let v1Type = (* ("note", "locals") *)
                                (if NameMap.mem v1name locals then
                                        (getType (NameMap.find v1name locals), "locals")
                                else if NameMap.mem v1name globals then
                                        (getType (NameMap.find v1name globals), "globals")
                                else raise (Failure ("fatal error")))
      in
        let v1RetType = getType var1 in
        let e1Type = getType e1 in
                                if v1RetType= "chord" && e1Type = "note" then
                                        (if snd v1Type = "locals" then
                                ((getChord (NameMap.find v1name locals)).notelist <- getNote
e1 :: (getChord (NameMap.find v1name locals)).notelist;
                                (NameMap.find v1name locals), (locals, globals))
                                        else if snd v1Type = "globals" then
                                ((getChord (NameMap.find v1name globals)).notelist <- getNote
e1 :: (getChord (NameMap.find v1name globals)).notelist;
                                (NameMap.find v1name globals), (locals, globals))
                                        else raise (Failure ("fatal error")))
                                else if v1RetType= "staff" && e1Type = "note" then
                                        (if snd v1Type = "locals" then
                                ((getStaff (NameMap.find v1name locals)).chordlist <-
{notelist=[getNote e1]} :: (getStaff (NameMap.find v1name locals)).chordlist;
                                (NameMap.find v1name locals), (locals, globals))
                                        else if snd v1Type = "globals" then
                                ((getStaff (NameMap.find v1name globals)).chordlist <-
{notelist=[getNote e1]} :: (getStaff (NameMap.find v1name globals)).chordlist;
                                (NameMap.find v1name globals), (locals, globals))
                                        else raise (Failure ("fatal error")))
                                else if v1RetType= "staff" && e1Type = "chord" then
                                        (if snd v1Type = "locals" then
                                ((getStaff (NameMap.find v1name locals)).chordlist <- getChord
e1 :: (getStaff (NameMap.find v1name locals)).chordlist;
                                (NameMap.find v1name locals), (locals, globals))
                                        else if snd v1Type = "globals" then
                                ((getStaff (NameMap.find v1name globals)).chordlist <- getChord e1 ::
(getStaff (NameMap.find v1name globals)).chordlist;
                                (NameMap.find v1name globals), (locals, globals))
                                        else raise (Failure ("fatal error")))
                                else if v1RetType= "part" && e1Type = "staff" then
                                        (if snd v1Type = "locals" then
                                ((getPart (NameMap.find v1name locals)).stafflist <- getStaff e1
:: (getPart (NameMap.find v1name locals)).stafflist;
                                (NameMap.find v1name locals), (locals, globals))
                                        else if snd v1Type = "globals" then
                                ((getPart (NameMap.find v1name globals)).stafflist <- getStaff
e1 :: (getPart (NameMap.find v1name globals)).stafflist;
```

```
                                        (NameMap.find v1name globals), (locals, globals))
                                                else raise (Failure ("fatal error")))
                                    else raise (Failure ("cannot perform: add(" ^ fst v1Type ^ ", " ^  e1Type ^
")"))
      | Call("print", [e]) ->
          let v, env = eval env e in
       (if getType v = "int" then
          print_endline (string_of_int (getInt v))
       else if getType v = "float" then
         print_endline (string_of_float (getFloat v))
       else if getType v = "bool" then
         print_endline (string_of_bool (getBool v))
       else
         print_endline(getType v));
       (Bool false), env
                | Call("play", [e]) ->
                        ignore (match e with
                Id(i) -> i
                        | _ ->  raise (Failure ("argument of play must be an identifier")));
                let e1, env = eval env e in
                        (if getType e1 = "part" then
                                        let p = getPart(e1) in
                                        (let start = (import_decl ^ class_start ^ staff_start) in
                                        let oc = open_out "Output.java" in
                                                (fprintf oc "%s" start;
                                        let print_note x =
                                                fprintf oc "%s\n" ("\nt.add(createNoteOnEvent(" ^
(string_of_int x.pitch) ^
                                                        ", tick," ^ (string_of_int x.intensity) ^ "));" ^
                                                        "\ntick += getDuration(" ^ (string_of_float
x.duration) ^ ");" ^
                                                        "\nt.add(createNoteOffEvent(" ^ (string_of_int
x.pitch) ^ ", tick));");
                                        in
                                        let print_chord x =
                                                fprintf oc "%s\n" ("\nt.add(createNoteOnEvent(" ^
(string_of_int x.pitch) ^
                                                        ", tick," ^ (string_of_int x.intensity) ^ "));" ^
                                                        "\nt.add(createNoteOffEvent(" ^ (string_of_int
x.pitch) ^ ", tick + getDuration(" ^ (string_of_float x.duration) ^ ")));");
                                        in
                                        let chord_iter y =
                                                (if List.length y.notelist < 2 then
                                                        List.iter print_note y.notelist
                                                else
                                                        List.iter print_chord y.notelist;
                                                        fprintf oc "%s\n" ("\ntick += getDuration(" ^
(string_of_float (List.hd y.notelist).duration) ^ ");");
```

```
                                                )
                                                in
                                                let staff_iter staff =
                                                        (fprintf oc "%s\n" ("\nt = createTrack(" ^
(string_of_int staff.instrument) ^ ");" ^ "\ntick = 0;");
                                                        List.iter chord_iter (List.rev staff.chordlist))
                                                in
                                                        List.iter staff_iter (List.rev p.stafflist));
                                        fprintf oc "%s" staff_end;
                                        fprintf oc "%s" (
                                                main_start ^ "new Output(" ^
                                                (string_of_int (p.bpm)) ^ ", " ^ (string_of_float
(p.beatsig)) ^
                                                ");");
                                        fprintf oc "%s" main_end;
                                        fprintf oc "%s" class_end;
                                        close_out oc)
                        else raise (Failure ("argument of play must be a part")));
                                (Bool false), env
                        | Call(f, actuals) -> (* check *)
                                let fdecl =
                                        try NameMap.find f func_decls
                                        with Not_found -> raise (Failure ("undefined function: " ^ f))
                                in
                                let actuals, env = List.fold_left
                                        (fun (actuals, env) actual ->
                                        let v, env = eval env actual in v :: actuals, env)
                                        ([], env) actuals
                                in
                                let (locals, globals) = env in
                                try
                                        let globals = call fdecl (List.rev actuals) globals
                                        in Bool false, (locals, globals)
                                with ReturnException(v, globals) -> v, (locals, globals)
                        in
                        (* Execute a statement and return an updated environment *)
                        let rec exec env = function
                                Block(stmts) -> List.fold_left exec env stmts
                                | Expr(e) -> let _, env = eval env e in env
                                | If(e, s1, s2) ->
                                        let v, env = eval env e in
                                        exec env (if getBool v != false then s1 else s2)
                                | While(e, s) ->
                                        let rec loop env =
                                                let v, env = eval env e in
                                                if getBool v != false then loop (exec env s) else env
                                        in loop env
                                | For(e1, e2, e3, s) ->
```

```
                        let _, env = eval env e1 in
                        let rec loop env =
                                let v, env = eval env e2 in
                                if getBool v != false then
                                        let _, env = eval (exec env s) e3 in
                                        loop env
                                else
                                        env
                        in loop env
                | Return(e) ->
                        let v, (locals, globals) = eval env e in
                if getType v = fdecl.rettype then
                                raise (ReturnException(v, globals))
        else raise (Failure ("function " ^ fdecl.fname ^ " returns: " ^ getType v ^ " instead of " ^
fdecl.rettype))
        in (* done with below *)
        (* call: enter the function: bind actual values to formal args *)
        let locals =
                try List.fold_left2
                        (fun locals formal actual ->
                        NameMap.add formal.paramname actual locals)
                NameMap.empty fdecl.formals actuals
                with Invalid_argument(_) ->
                        raise (Failure ("wrong number of arguments to: " ^ fdecl.fname))
        in
        let locals = List.fold_left (* Set local variables to 0 *)
                (fun locals local -> NameMap.add local.varname (initIdentifier local.vartype)
locals)
                locals fdecl.locals
        in (* Execute each statement; return updated global symbol table *)
                snd (List.fold_left exec (locals, globals) fdecl.body)

        (* run: set global variables to 0; find and run "main" *)
        in let globals = List.fold_left
                (fun globals vdecl -> NameMap.add vdecl.varname (initIdentifier vdecl.vartype)
globals) (* vdecl is the identifier, "X" *)
                NameMap.empty vars
        in try
                call (NameMap.find "main" func_decls) [] globals
        with Not_found ->
                raise (Failure ("did not find the main() function"))
```