mTunes: A Midi Music Development Language

HeeWook Lee Sharon Price[1] Hideki Sano Huitao Sheng
{hl450, sbp2001, hs2160, hs734}@columbia.edu

[1] Group Leader

Table of Contents

1 White Paper

1.1 Introduction

Any music enthusiast who wants to create his or her own midi songs typically has two choices: professional audio composition and editing software; and Sun's Java midi package (javax.sound.midi). For a small fee ranging from $100 to over $5000, professional/amateur audio composition and editing software can be purchased. For a small time commitment ranging from 100 to over 5000 hours, any Java programmer can become fluent in Java midi. Alas, what is the midi enthusiast to do?

mTunes! The solution to all of life's (midi) problems! A free, open source language that allows users to not only compose in midi, but automatically generate songs using Functions and Procedures. No longer will musicians have to manually place every note on every track using a graphical interface, but instead they are now free to specify a note once and easily use that note anywhere in a song. The extra level of abstraction from Java midi means a pleasant and simple user experience that allows creativity to blossom.

1.2 mTunes Description

mTunes: An easy-to-use, object-oriented, portable, reusable, midi composition language. Allows users to add and remove instruments, entire tracks, and single notes, and perform any transformation that they desire.

1.2.1 Easy-To-Use

Music composers already have to deal with hundreds of musical commands. mTunes has a small, simple command set that is easy to master. Instead of having composers who are not computer-savvy trying to use foreign interfaces, they only need to remember a few easy keywords and their associated parameters. This allows composers to focus on their music instead of on overly complicated software.

1.2.2 Object-Oriented

mTunes composers create specify what instrument they want to be associated with a particular Instrument (the mTunes version of a track). Notes, and groups of Notes called Sequences, are created in a manner similar to instantiating variables in Java and C. Once created, different parameters of the individual Notes and the Notes in a Sequence can be accessed for transformation. Also, both Notes and Sequences can be

added to Instruments.

1.2.3 Portable

The interpreter is written in Java and based on Java music packages, so it can be used on almost every operating system and architecture still in use today. This provides mTunes with seamless portability. The mTunes code itself is written in a text file, which enables it to be transferred to any other computer.

1.2.4 Reusable

Even with an easy to use language, reusability of code is important; users can always spend extremely long periods of time coding to produce intricate programs. With mTunes, previously written code, with little or no changes, can be used to create new songs.

1.3 Language Features

1.3.1 Data Types

For simplicity, there are only three basic data types available for users of mTunes – *Notes*, *Sequences*, and *Instruments*. Notes represent standard musical notes, and Sequences are series of Notes, in the order in which they are played. Instruments are very similar to tracks, in that they get set to a particular kind of musical instrument, then have Notes and Sequences added to their queues. For computational purposes, there are also *Numbers*, which are equivalent to Java's double.

1.3.2 Basic Commands

The most important three commands are: *save*, *playSong*, and *add*. The save command tells mTunes where to save the midi file, and playSong indicates that the song should be played aloud, instead of just saved. add is used to append Notes and Sequences to the end of an Instrument's queue.

1.3.3 Flow Control

To allow for repetition and automatically generating Notes and Sequences, for and foreach loops have been included in mTunes. The for loop iterates the number of times specified, while the foreach loop iterates through every Note in the specified sequence.

1.4 Example Syntax

1.4.1 Sample comment

Note: Everything after the ; is ignored in a line

;<comment>

```
;this is a comment! YAY!
```

1.4.2 Creating a Note and Sequence

<type> <identifier> = <value>;

```
Note aNote = C#
Sequence aSeq = C aNote D
```

1.4.3 Saving the Song as a .midi File

save <location>

```
save "awesomesong.midi"
```

1.4.4 Playing a Song After Compilation

playSong

```
playSong
```

1.4.5 Creating a for or foreach Loop

for (<starting number>, <ending number>, <counter>) {

    <statements>

}

```
for(2, 14, current) {
      oneNote.pitch = current + oneNote.pitch
}
```

-or-

foreach(<Sequence>, <current Note identifier>) {

    <statements>

}

```
foreach(thisSequence, currentNote) {
      currentNote.length = currentNote.length * 6
}
```

1.4.6 Creating a Function or Procedure

<Function/Procedure> <identifier> (<parameters>) {
    <statements>
}

```
Procedure awesome(Sequence poke) {
      foreach(poke, thisNote) {
            thisNote.volume = 0.5 * thisNote.volume
      }
}
```

2 Tutorial

2.1 How to run

To compile and run, at the command prompt simply type:
$mtunesc.exe <filename>

2.2 Order of Statements

An mTunes file has to follow a certain order for declaring certain statements. The “save” statement always has to come in the beginning of the file if you wish to save the file as midi-formatted file and “playSong” statement has to come at the end of the file if you would like the computer to play the sound upon executing the code. Other statements can be used anywhere else.

2.3 Basics

Before we go on, you should keep in mind that every variable, function, or procedure has to be declared before they are used.

2.3.1 Note Definition

Note is one of the most basic data types that you will have to deal with since we are all about composing music.

If you would like to define a note variable with pitch C in octave 4, with length 0.25(which is the quarter note). You would have to declare it as:

```
Note note1 = C4_0.25
```

Sharp and flat are expressed with ‘#’ and ‘$’ respectively. So if you want C# in octave 4 with length 0.25. You would declare it as:

```
Note note1 = C#4_0.25
```

The underscore and length can be left out and it will be automatically set the length to the default length of 0.25.

Since pitch is internally defined as integer, we can perform mathematical

operation on the pitch. All the mathematical operations are done by accessing the attributes of the notes through the dot operator.

2.3.2 Sequence Definition

Sequence is one of the most powerful data types used in mTunes. A Sequence is basically an array of notes, declared as:

```
Sequence s1 = C5_0.5 D_0.5 E_0.5 F_0.5 G_0.5 A_0.5 B_0.5 C6_0.5
```

If s1 is played, it will play starting from C5_0.5 sequentially to C6_0.5, which is basic middle C major scale.

2.3.3 Instrument Declaration

mTunes support all the standard java MIDI instruments. There are 16 channels available for use, which means at most 16 instruments can play sound at the same time. Instrument variables are previously defined for users since there can only be 16 instances, I1 through I16. If you want to set Instrument 1 to PIANO, you would do the following:

```
I1 = PIANO
```

2.3.4 Note attributes.

There are four note attributes for each Note, which are octave, length, pitch, and volume. After a note is declared, each attribute can be accessed using the dot operator. For example, if n1 is a Note variable, to change the length of n1 to 0.75, the code would look like the following:

```
n1.length = 0.75
```

If you want to change the pitch to F#:

```
n1.pitch = F#
```

2.3.5 Sequence Operations

Many operations can be applied to an mTunes Sequence. If you want to concatenate two sequences, s1 and s2, you will need to write code like this:

```
s1 = s1 + s2
```

Also, the foreach loop can be applied to a Sequence but we will get into that in the latter section of this tutorial.

2.3.6 Numbers

Numbers in mTunes are done in IEEE 32bit-floating point number. If we have to take a number as an argument for a function we would declare as the following:

```
Number variableName
```

2.4 Flow Control

In mTunes, flow control is done by two statements, for and foreach.

If you want to iterate 10 times using the for loop, you would declare the for statement as following:

```
for(1,10, count){
      statements…
}
```

Note that both index bounds are inclusive so it goes from 1 to 10. Count is a variable that will be accessible during the for loop, which can be named anything legal. It stores the number of the current iteration. So during the ith iteration count will be i.

The foreach statement is often used for sequence manipulation. This statement takes two arguments – a Sequence and a variable name for the current Note. If you have a sequence of 10 notes, and you would like to change the double the length of each Note, you would do the following:

```
foreach(targetSequence, aNote){
      aNote.length = 2*aNote.length
}
```

Note that aNote has to be declared before the foreach statement is used. For each iteration it will access the length and multiply by 2. aNote can be any Note desired. aNote will hold the instance of the note for each iteration of foreach statement.

2.5 Function Declaration

Function is often useful to use when there are repetitive operation that you need to perform. There are two types of function in mTunes, function and procedure. Functions and procedures have to be declared before it's used in the file since the file is executed sequentially

2.5.1 Function

Functions can return a sequence or note at the end of the function execution. For example, given a number i and sequence s1, if you want to repeat the sequence s1 i times and return the resulting sequence, your code would look something like this:

```
function repeat(Number times, sequence block){
	Sequence result
	Number count
	for(1,times,count){
		result = result + block
	}
	return result
}
```

2.5.2 Procedure

Procedures do not return anything - they just executes the statements in the body sequentially unless there is a flow control statement. If we were to write the function repeat from section 2.5.1 as a procedure, we would write it like this:

```
procedure repeat(Number times, sequence block){
	Sequence temp = block
	Number count
	for(1,times,count){
		block = block + temp
	}
```

```
}
```

2.6 Listening and Saving Music

If you want to play the song you have composed in mTunes, all you have to do is put the playSong command on the last line of the file. Upon execution, Java MIDI playback will play the file. Before the song is played, the score will be displayed in a seperate window that will appear.

In order to save the music in a midi file called, “Test.midi”, you will need to give the following command on the first line of the mTunes file.

```
save “Test.midi”
```

Upon execution, the file will be saved in the working directory.

2.7 Example: C major scale

Playing a middle C major scale can be done in many different ways but here we provide one example:

```
;Playing C major scale example
save “Cmajor.midi”
I1 = PIANO
Sequence cMajor = C4 D4 E4 F4 G4 A4 B4 C5
add I1 cMajor
playSong
```

This program simply plays the middle C major. Sequence is set with the notes for the scale in order then it is added to I1 which is set as PIANO. Then it just simply plays the scale.

2.8 Example: Chromatic scale

```
save “chromatic.midi”
I1 = PIANO
Number n1
```

```
Sequence s1
for(60, 72, n1){
      Note note1
      note1.pitch = n1
      s1 = s1 + note1;
}
add I1 s1
playSong
```

This program simply plays the chromatic scale starting from middle C. This program demonstrates how mathematical operation can be applied to the note attributes. Internally each half step in pitch is a difference of one in music, so from C to D are a whole note away and you will have to add 2 to get to D. Since the chromatic scale plays every possible note in an octave sequentially, every single time it loops through, pitch will increase by 1 (half-step in music lingo).

2.9 Example: Fibonacci scale

```
function fibonacci(Number first, Number second){
      Number fib = first + second
      return fib
}
Sequence s1
Number prev2 = 0
Number prev1 = 1
Number current
for (0, 50, n1) {
      Note note1
      Number tmp
      if (n1 < 2) {
            if (n1 == 0) {
                  note1.pitch = prev2 + 60
            }
            else {
                  note1.pitch = prev1 + 60
            }
```

```
	}
	else {
		current = fibonacci(prev2, prev1)
		prev2 = prev1
		prev1 = current
		tmp = current % 12
		note1.pitch = tmp + 60
	}
	s1 = s1 + note1
}
I1 = PIANO
add I1 s1
playSong
```

This program plays a scale based on the Fibonacci numbers. But since Fibonacci numbers are increasing rapidly, we perform mod 12 on every Fibonacci number calculated since there are only 12 pitches in each octave. So this program will play notes based on Fibonacci sequence in the fourth octave. 60 stands for middle-C

2.10 Example: Jingle Bells

```
save "jinglebell.midi"
I1 = PIANO
I2 = STEEL_DRUMS
function repeat(Number times, Sequence block){
	Sequence result
	for(1,times,count){
		result = result + block
	}
	return result
}
function transpose(Number howmanyOctave, Sequence target){
	Note current
	foreach(target, current){
		current.pitch = current.pitch + 12*howmanyOctave
	}
```

```
      return target
}
Sequence soprano
Sequence bass
Sequence temp = A4_0.25 A4_0.25 A4_0.5
temp = repeat(2,temp)
Sequence  temp2  =  A4_0.25  C5_0.25  F4_0.375  G4_0.125  A4_1  B$4_0.25
B$4_0.25 B$4_0.375 B$4_0.125 B$4_0.25
temp = temp + temp2
Sequence temp3 = A4_0.25 A4_0.25 A4_0.125 A4_0.125
temp = temp + temp3
Sequence temp4 = A4_0.25 G4_0.25 G4_0.25 A4_0.25 G4_0.5 C5_0.5
Sequence firstpart = temp + temp4
Sequence temp5 = C5_0.25 C5_0.25 B$4_0.25 G4_0.25 F4_1
Sequence secondpart = temp + temp5
soprano = firstpart + secondpart
Sequence temp6 = F2_0.25 C3_0.25 C2_0.25 C3_0.25
temp6 = repeat(2,temp6)
Sequence  temp7  =  F2_0.25  C3_0.25  B$1_0.25  D2_0.25  F2_0.25  F2_0.25
E2_0.25 E$2_0.25
Sequence  temp8  =  D2_0.25  F2_0.25  B$1_0.25  F2_0.25  F1_0.25  F2_0.25
C3_0.25 F2_0.25
Sequence repeatingBass = temp6 + temp7 + temp8
Sequence firstEnding = C2_0.25 G2_0.25 G1_0.25 G2_0.25 C2_0.25 C2_0.25
D2_0.25 E2_0.25
Sequence  secondEnding  =  C2_0.25  C2_0.25  D2_0.25  E2_0.25  F2_0.25
C2_0.25 F2_0.5
bass = repeatingBass + firstEnding + repeatingBass + secondEnding
add I1 soprano
add I2 bass
Sequence transPosedSop = transpose(1, soprano)
Sequence transPosedBass = transpose(1, bass)
addI1 transPosedSop
addI2 transPosedBass
playSong
```

This example just demonstrates that mTunes actually can play a song everyone knows. Many concatenations of sequence used and repeat function used for the repeating melody. This program also uses two instrument variables, which means two instruments will be playing at the same time.

3 Language Reference Manual

3.1 Lexical conventions

3.1.1 Comments

mTunes supports one-line comments only. All comments start with ";" and continue through the end of the line. Comments must be on their own lines.

3.1.2 Identifiers (Names)

An identifier is an alphanumeric sequence. The first character must be a lower-case letter, while the remaining characters can be upper- and lower-case letters, numbers, and the "_" symbol. Upper and lower case letters are considered different.

3.1.3 Reserved Words

The following identifiers are reserved words in mTunes and may not be used otherwise:

```
function
procedure
for
foreach
print
add
save
if
else
break
return
```

3.1.4 Numbers

Numbers are denoted in decimal format, consisting of digits and an optional decimal point. mTunes does not allow the use of numbers starting with decimal point. For example, .25 will not be accepted as a number. All numbers, whether integer or decimal, will be internally represented as 32-bit IEEE floating point numbers except when mTunes is doing mod operation.

3.1.5 Other Tokens

The symbolic characters supported by mTunes are:

```
{ } ( ) , ;
+ - * / % #
= += -= *= /=
>= <= > < == !=
&& || !
```

3.2 Types

The following data-types are available for use in mTunes:

- Note – a musical note containing information about pitch, duration, volume, and octave
- Sequence – a series of Notes, internally represented as an array of notes
- Instrument – the type of musical instrument used to play either a Sequence or Note(s)
- Function/Procedure – a user-defined method for generating a Sequence or Sequences
- Number – 32-bit IEEE floating point number

3.2.1 Variable Declaration

All data-types are considered to be variables Once an identifier is set to a user-defined Function, the identifier cannot be used to create a new Function. Since Functions can be accessed from anywhere inside the program, it would be impossible to determine which version of the Function should be used. All variables, functions, and procedures have to be declared before they are used.

For all variables except instruments, the following format is used:

```
<type> <identifier>
```

-or-

```
<type> <identifier> = <expression or value>
```

Notes and Numbers can be used without assigning them to a variable name (hard-coded by the user into the program). If a value is not assigned to a variable during declaration, it cannot be used until a value is assigned.

There are sixteen predefined instrument variables available for use. The user cannot create any new instrument variables, but can change the value assigned to any given instrument. They are denoted “I<1-16>”. To set the type of instrument:

```
I<1-16> = <instrument name>
```

There are almost two-hundred predefined instruments in mTunes, almost every instrument supported by midi. The full list is included in Appendix A.

3.2.2 Scope

All variables created within a Function or loop are local to that Function or loop. If an instrument variable is set within a Function, it remains set to that instrument until set otherwise. Any variable created outside of any control-flow statement is considered to have global scope.

3.3 Expressions

3.3.1 Primary Expressions

Primary expressions consist of combinations of identifiers, constants, function calls, and any other expression surrounded by “(“ and “)”.

3.3.2 Identifiers

An identifier is an expression that always evaluates to the value bounded to it.

3.3.3 () Expression

A parenthesized expression is a primary expression which evaluates to the value of the enclosed expression. The parentheses are used to insure that an expression evaluates to the correct value, despite precedence rules.

3.3.4 Arithmetic Expressions

Arithmetic expressions take primary expressions as operands, with the exception modulo, which can only take primary expressions that evaluate to numbers. The operators “+”, “-“, “*”, “/”, and “%” indicate addition, subtraction, multiplication, division, and modulo. Multiplication, division, and modulo have the highest precedence, then addition and subtraction.

3.3.5 Relational Expressions

Relational expressions can only be used with if statements and take primary expressions as operands. The binary relational operators “>=”, “<=”, “==”, “!=”, “>”, and “<” are used to test whether the first operand is greater than or equal to, less than or equal to, equal to, not equal to, greater than, or less than the second operand. If the expression evaluates to true, the body of the if statement is executed.

3.3.6 Logical Expressions

Logical expressions can only be used with if statements and take relational expressions as operands. The logical operators “&&”, “||” are used to test whether the first operand and, or the second is true. The “!” operator can be used on just one operand to test if the operand evaluates to false, or if the first “!=” the second. If the expression evaluates to true, the body of the if statement is executed.

3.4 Statements

Statements in mTunes must be entirely on one line. Everything on any given line, until a “;” or a newline character is reached is considered to be one statement. Unless there is a comment on a line, statements do not have to terminate with any symbol. Multi-line statements are not supported. A sequence of statements will be executed sequentially, unless the flow-control statements indicate otherwise.

3.4.1 Assignment

All variable assignments take the form:

```
<variable name> = <value or expression>
```

3.4.2 If Statement

Conditional statements use either the “if” keyword or the “if” and “else” keyword. “else if” is not supported. Conditional statements take the form:

```
If (<logical expression>) {
      <statements>
}
```

-or-

```
If (<logical expression>) {
      <statements>
```

```
}
else {
      <statements>
}
```

If the first logical expression evaluates to true, the statements in the "{ }" immediately following the if are executed. If the logical expression evaluates to false and there is an else clause, the statements inside the "{ }" immediately following the else are executed.

3.4.3 For Statement

There is one basic type of loop in mTunes, the for statement.

```
for (<starting Number>, <ending Number>, <identifier>) {
      <statements>
}
```

Both the starting Number and ending Number will be interpreted as integers, though they can be Number variables or floating point numbers. If a non-integer number is used, the floor of that number will be used instead. The identifier is the variable that will be assigned the number of the current iteration for the life of the loop. The variable is assigned the starting number for the iteration, then incremented by one for each subsequent iteration. The last iteration occurs when the variable is equal to the ending value.

3.4.4 Break Statement

The break statement will automatically end the inner-most for loop in which it is located. The statement simply consists of the keyword "break".

3.4.5 Foreach Statement

There is one basic type of loop in mTunes, the for statement.

```
foreach (<sequence>, <identifier>) {
      <statements>
}
```

The Foreach statement loops through every note in the sequence, starting with the first note that was added to the sequence. The identifier is set to the current note in the sequence, and all of the elements in the note can be accessed through the regular notation defined in Section 3.8. The identifier should be declared as a Note type variable before using foreach statement.

3.4.6 Function Call

Functions, both user-defined and included, are called the same way:

```
<identifier>(<parameter1>, <parameter2>, ...)
```

3.4.7 Return Statement

User-defined functions can return a Note, Sequence, and Number. which is accomplished through the return statement. If a user-defined function does not require a Sequence, Note, or Number to be returned, procedure should be used rather than function.

```
return <sequence name or series of notes>
```

3.5 Function Definition

Functions in mTunes cannot be overridden, nor can they be nested. All function arguments are passed by value. A user-defined Function must take the form:

```
procedure <identifier> (<type1 parameter1>, <type2 parameter2>, ...) {
      <statements>
}
```

-or-

```
function <identifier> (<type1 parameter1>, <type2 parameter2>, ...) {
      <statements>
      <return statement>
}
```

Procedure creates a new function that does not return anything, while function returns a Sequence, Note, or Number. The return statement must appear at the end of every function declaration. The identifier is the name of the function. There can be zero

or more parameters to a function, comma separated if there is more than one. The type of the parameter followed by a space must precede the parameter name.

3.6 Internal Functions

3.6.1 Console Output

To print to the console, the Print commands are used.

```
print <strings>
```

The strings must be surrounded by double quotes. Print automatically includes a new line after the output has been printed.

3.6.2 Save

To save the music generated by a program as a midi file, the save function is used.

```
save <path>
```

The path includes the desired file-name for the file, relative to the directory from which mTunes is run. The path must be surrounded by double quotes. The save command must be the first line of the program, unless the playSong command is also being used, in which case the save command must be the next-to-last line. If save is not on the first line, program will not run and the compiler will throw an Unexpected token error.

3.6.3 Add

To add a finalized Sequence or Notes to an instrument's queue:

```
add I<1-16> <Sequence or Notes>
```

There can only be either one Sequence or a series of Notes used with the add command. To add multiple Sequences, multiple add commands must be called. When the final song is played, each instrument with Notes in its queue plays through its queue simultaneously from the start of the song. Each instrument plays until its queue is empty. If the kind of instrument playing is changed at some point in the queue, the

change will affect only the contents of the queue that occur after the change. Before using add statement, I<1-16> has to be initialized with an Instrument, otherwise the compiler will throw an error.

3.6.6 PlaySong

The “playSong” command tells the compiler to play the generated song after compilation. This command must be the last line of the program, otherwise the compiler will throw an error. Before playing the song, there will be GUI version of music score in music-sheet format.

3.7 Note Definition

3.7.1 Pitch

The pitch of a note is denoted by “C”, “D”, “E”, “F”, “G”, “A”, “B”, and “R”. “R” is a rest, while the rest of the pitches correspond to the standard pitches on the staff. The pitch is the only part of a note that must be explicitly defined. All other parts are optional, in which case they automatically default to the current default (if applicable).

3.7.2 Octave

There are ten available octaves, numbered 0 through 9. The default is 4 for both individual notes and globally, which includes middle-C. To change the octave for one note, the letter representing the pitch is preceded by a number indicating the desired octave.

3.7.3 Sharp and Flat

Sharp is denoted by “#” and flat by “$”. If the note is to be sharp or flat, the pitch is preceded by the symbol.

3.7.4 Duration

The default duration is a quarter note (internally represented as 1), though any length can be specified. The shortest supported duration is 0. The duration follows the pitch, and the sharp or flat symbol if present. It can be an integer variable or any number as defined above.

3.7.5 Volume

The default default volume is 100. Number used to set volume will be rounded

down to a decimal number. 0 volume indicates mute. To change the volume for one note, the pitch (and the duration, if applicable) ".volume" command in 3.7.6 must be used.

3.7.6 Accessing Note Attributes

Once a Note is created, its attributes can still be accessed and changed:

```
<Note>.pitch
<Note>.length
<Note>.volume
```

The first command returns the pitch of the Note, as an integer, from 0 to 120, with 0 being a rest. The second command returns the duration of the Note, and the last command returns the volume of the Note. All three commands return integers, except for length, which can be floating point, and can be used anywhere a Number would be used.

Any of the attributes can be set equal to a new value:

```
<Note>.<attribute> = <value>
```

The value consists of any valid Number (as described above for each attribute). For pitch, a valid letter and sharp/flat combination, or the equivalent numerical value, can be used.

3.8 Sequence Definition

Sequences consist of one or more Notes to be played in the order that they appear in the Sequence. They can be built up from existing Sequences and individual Notes.

3.8.1 Creation

To create a new Sequence:

```
Sequence <identifier> = <value>
```

The value consists of space delimited Notes and Sequences.

3.8.2 Adding Additional Notes and Sequences

Once a sequence is created, both Notes and Sequences can be appended to the beginning and end of a Sequence without the use of a foreach loop.

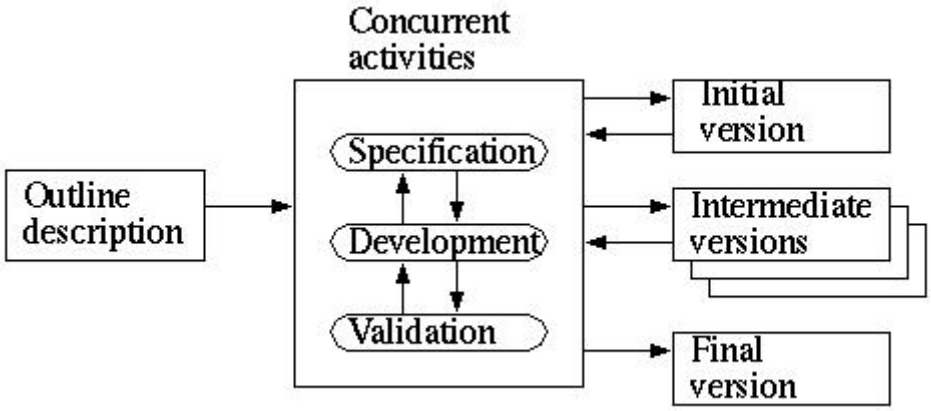
```
<identifier> = <identifier1> + <identifier2>
```

Sequence can be concatenated with '+' operator. Resulting sequence, when it's played, it will play the contents of identifier1 first then identifier2. Identifier1 and identifier2 can be either Notes or Sequences.

4 Project Plan

4.1 Process Used for Planning, Specification, Development, and Testing

Our group followed the evolutionary development model of software development. First we came up with our original description of our language. We then proceeded to go through many versions of the language specifications as we wrote our language reference manual, received feedback from the T.A., and wrote our code. The intermediate versions, and especially the final version, underwent extensive testing.



4.2 Programming Style Guide

4.2.1 Java code

- Parenthesis should hug the code inside them
- One space should proceed and follow all operands
- No space should appear between function calls and the following parenthesis
- Indent by one tab for each new level
- The left brace should be at the end of the line of the statement that requires the bracket
- The right brace should be on its own line, indented to the level of the opening statement
- Braces do not have to be included if not needed by compiler, in which case the following line should still be indented by one more tab
- All file names should begin with MTunes, except the executable, which should be entirely lower case
- Variable names should be appropriate and indicate what they are

- For the purposes of adding code to the final report, there is a limit of 60 characters per line

4.2.2 ANTLR code

- All token names should be entirely upper case
- All non-token names should be entirely lower case, '_' delimited
- One line lexer rules should have the ':' indented by two tabs
- All rules should be formatted as follows: name on first line; two tabs, ':', space, rule
- two tabs, '|', space, rule for all remaining rules, each on own line
- two tabs, ';', alone on last line
- One blank line should appear between rules
- For the purposes of adding code to the final report, there is a limit of 60 characters per line

4.2.3 mTunes test code

- File name should be test<#>.mt

4.3 Project Timeline

4.3.1 Original Timeline

- September 16 – Good idea that everyone can live with for language
- September 20 – White paper finished
- October 13 – Language Reference Manual completed enough to start on lexer and parser
- October 20 – Language Reference Manual finished
- October 23 – Lexer and parser working
- November 14 – Tree walker and libraries working
- November 30 – Enough testing completed to be confident that compiler works properly
- December 11 – Final report completed (excluding code and logs)
- December 17 – Final working version of compiler finished, final report finished

4.3.2 Actual Timeline (Project Log)

- September 11 – Good idea chosen – Java-based midi language
- September 22 – White paper finished

- October 14 – First grammar finalized
- October 16 – Lexer written
- October 27 – First Language Reference Manual finished
- October 30 – Parser written
- November 4 – Work began on version 2.1
- November 12 – Version 2.1 Language Reference Manual finished
- November 28 – Version 2.1 lexer worked
- November 30 – Version 2.1 parser worked
- December 1 – Testing of lexer and parser began
- December 4 – Tree walker coding began
- December 11 – Final report began
- December 13 – Tree walker worked
- December 13 – Rigorous testing began
- December 16 – Executable written to run compiler
- December 17 – Final working version of compiler finished, final report finished

4.4 Roles and Responsibilities

Originally Sharon and HeeWook were in charge of documentation, learning jMusic, and coding the tree walker. Huitao and Hideki were in charge of the lexer, parser, and any non-jMusic libraries that were needed. However, as the semester evolved, so did our roles. The final break down is as follows:

- Sharon – group leader, documentation, basic jMusic, lexer, parser, executable
- HeeWook – documentation, advanced jMusic, lexer, parser
- Huitao – tree walker, libraries, testing
- Hideki – tree walker, libraries, testing

Every member was required to send an email reporting any changes that they made with the new code as an attached file whenever they worked on the code. Also, we had group meetings, varying from once a week towards the beginning of the semester to two and more times a week towards the end.

4.5 Software Development Environment

4.5.1 Code

HeeWook, Huitao, and Hideki all coded on Windows machines. HeeWook used UltraEdit, a text editor. Huitao used EditPlus, another text editor, and Hideki used Eclipse, IBM's Java IDE. HeeWook and Huitao used the Windows command prompt (cmd.exe) to compile and run code. Sharon coded on a Linux machine running Slackware 9.1. She used Emacs to write, and the command line to compile and run.

All code was written in either Java (compiled with Java 1.4.x) or ANTLR. The lexer, parser, and tree walker were written entirely in ANTLR, while the additional libraries were written in Java. The executable was also written in Java, and compiled using gcj and mingw on Cygwin 1.5.5.

4.5.2 Documentation

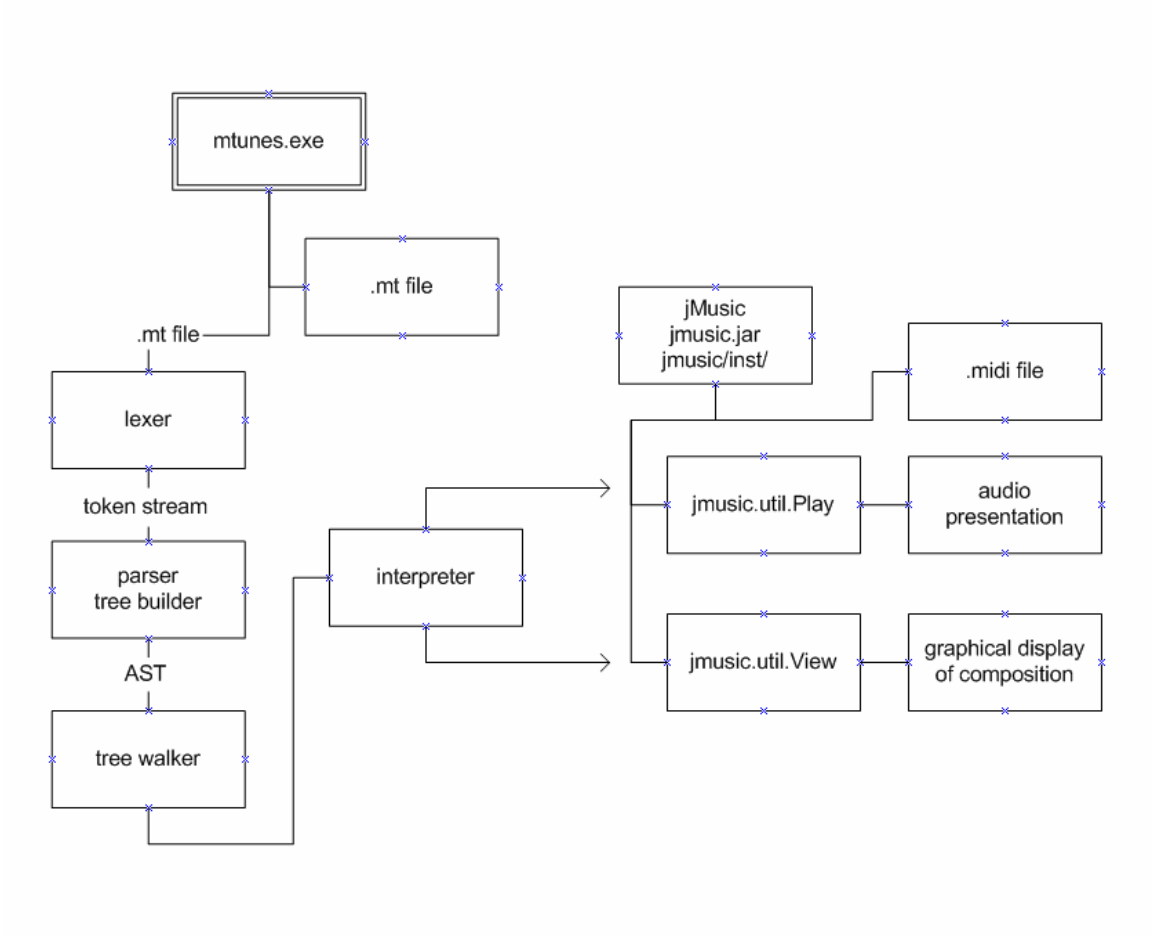
All documentation was written on Windows machines, using Microsoft Word almost exclusively. The flow chart was created using Microsoft Visio and imported into Word. The code appearing in the appendix was opened using Eclipse and copied into Word. This allowed code to remain properly formatted when copied into word. The final report was converted to a pdf using Adobe Acrobat 6.0 Professional.

4.5.3 Versioning System

Our group never set up an official versioning system. This came about as work on the code started and group members automatically began sending out the code that they had written or updated to all other members. While not the most efficient versioning system, email proved more than sufficient for our needs, as well as accessible from almost anywhere. As an added precaution, Sharon kept backup folders for each new version of the code.

5 Architectural Design

5.1 Major Components of Translator



5.2 Interfaces Between Components

The main file, mTunes.java, creates an instance of the lexer, MTunesLexer.java, parser, MTunesParser.java, and tree walker, MTunesWalker.java. The lexer creates a stream of tokens that get passed to the parser. The parser generates an abstract syntax tree (AST) that gets passed to the tree walker. The tree walker traverses the AST, executing code as it walks. The tree walker also creates a new instance of MTunesInterpreter.java. The interpreter calls functions from the jMusic library to create the midi file, display the staff and the notes, and to play the song.

5.3 Credits

- Lexer[1] – Sharon and HeeWook
- Parser[2] – Sharon and HeeWook
- Tree Walker – Huitao
- Interpreter – Huitao, Hideki
- Library – Hideki, Huitao, Queensland University of Technology, Australia
- Executable – Sharon
- jMusic – Sharon and HeeWook
- Testing – Hideki, Huitao

[1] Huitao wrote the original lexer, but it had to be completely rewritten.

[2] Hideki wrote the original parser.

6 Test Plan

6.1 Sample Test Files

6.1.1 fibonacci.mt

```
save "fibonacci.midi"

function fibonacci(Number first, Number second){
	Number fib = first + second
	return fib
}
Sequence s1
Number prev2 = 0
Number prev1 = 1
Number current
for (0, 50, n1) {
	Note note1
	Number tmp
	if (n1 < 2) {
		if (n1 == 0) {
			note1.pitch = prev2 + 60
		}
		else {
			note1.pitch = prev1 + 60
		}
	}
	else {
		current = fibonacci(prev2, prev1)
		prev2 = prev1
		prev1 = current
		tmp = current % 12
		note1.pitch = tmp + 60
	}
	s1 = s1 + note1
}
```

```
I1 = PIANO
add I1 s1
playSong
```

6.1.2 fibonacci.midi

6.1.3 jinglebell.mt

```
save "jinglebell.midi"
I1 = PIANO
I2 = STEEL_DRUMS
function repeat(Number times, Sequence block){
      Sequence result
      for(1,times,count){
            result = result + block
      }
      return result
}
function transpose(Number howmanyOctave, Sequence target){
      Note current
      foreach(target, current){
            current.pitch = current.pitch + 12*howmanyOctave
      }
      return target
}
Sequence soprano
Sequence bass
Sequence temp = A4_0.25 A4_0.25 A4_0.5
temp = repeat(2,temp)
Sequence  temp2  =  A4_0.25  C5_0.25  F4_0.375  G4_0.125  A4_1  B$4_0.25
B$4_0.25 B$4_0.375 B$4_0.125 B$4_0.25
temp = temp + temp2
Sequence temp3 = A4_0.25 A4_0.25 A4_0.125 A4_0.125
temp = temp + temp3
Sequence temp4 = A4_0.25 G4_0.25 G4_0.25 A4_0.25 G4_0.5 C5_0.5
Sequence firstpart = temp + temp4
```

```
Sequence temp5 = C5_0.25 C5_0.25 B$4_0.25 G4_0.25 F4_1
Sequence secondpart = temp + temp5
soprano = firstpart + secondpart
Sequence temp6 = F2_0.25 C3_0.25 C2_0.25 C3_0.25
temp6 = repeat(2,temp6)
Sequence temp7 = F2_0.25 C3_0.25 B$1_0.25 D2_0.25 F2_0.25 F2_0.25
E2_0.25 E$2_0.25
Sequence temp8 = D2_0.25 F2_0.25 B$1_0.25 F2_0.25 F1_0.25 F2_0.25
C3_0.25 F2_0.25
Sequence repeatingBass = temp6 + temp7 + temp8
Sequence firstEnding = C2_0.25 G2_0.25 G1_0.25 G2_0.25 C2_0.25 C2_0.25
D2_0.25 E2_0.25
Sequence secondEnding = C2_0.25 C2_0.25 D2_0.25 E2_0.25 F2_0.25
C2_0.25 F2_0.5
bass = repeatingBass + firstEnding + repeatingBass + secondEnding
add I1 soprano
add I2 bass
Sequence transPosedSop = transpose(1, soprano)
Sequence transPosedBass = transpose(1, bass)
addI1 transPosedSop
addI2 transPosedBass
playSong
```

6.1.4 jinglebell.midi

6.2 Test Suite

We used “white-box” testing to check our language for errors. Instead of just giving the compiler test files and trying to figure out what went wrong based on the output, almost everything that the lexer and parser do was printed to the screen and a graphical representation of the AST was displayed. The lexer printed out every token that it identified, along with additional information provided by print statements inside grammar.g. The parser printed out every statement that it identified in the same fashion, then displayed the GUI. This allowed for much more efficient debugging since it was much easier to track down the source of errors with so much available information.

6.3 Reasoning Behind Test Cases

Close to eighty test files were written during the semester, some more impressive than others. In general, test files were written to test one or two specific aspects of the language. Included above are some of the more interesting test cases and the midi files that they produce. They also show almost every available feature in mTunes.

6.4 Automation

Due to the nature of our language, there really was no useful way to automate testing. When testing the lexer and parser, the AST needed to be checked by hand since it was displayed in a gui form. Later, once the whole compiler had been put together, the only way to test the output was by listening to the audio output.

6.5 Credits

The majority of the testing was performed by Hideki, with second place going to HeeWook. However, all members wrote test cases and tested code as they wrote it.

7 Lessons Learned

7.1 Sharon

- There is always some error that we didn't find in the parser.
- No matter how hard you try to get things done early, they will not be.
- Documentation is harder to write than it looks.

7.2 HeeWook

- Everybody in the group needs to be constantly aware of what is being done by other group members.
- Documentation takes a lot longer to write than you always think it will.
- Having a female in the group is good – at least one female.

7.3 Huitao

- The group should have had more meetings.
- A real version control system should have been used instead of just emailing files back and forth.
- I should have had a notebook to track changes – it would have made life a lot easier.
- We should have decided on a more realistic syntax from the start instead of having to make many changes.

7.4 Hideki

- When doing a big group project, it's important to find and fix bugs earlier, otherwise they cause much worse problems later.
- It's necessary to constantly stop and just test the code to make sure everything works.
- The next part of the compiler is easier to code and get working if the previous part has been thoroughly tested instead of assumed to work.

7.5 Advice for Future Groups

When designing your language, come up with some very concrete ideas that you know you can definitely do without a significant amount of trouble from the very beginning. The main problem experienced by our group was creating a fairly vague language that had to be redefined and changed four or five times. This resulted in a lot of wasted effort and time. We were forced to spend a lot more time trying to finish everything during the last couple weeks instead of checking test cases and fixing minor bugs.

Despite thinking that our language was relatively simple, we still were not able to implement everything that we had intended to in the final version of our language. Even if something in your language seems simple and obvious, it very well might be the most difficult thing in the world to actually implement, so be prepared to not be able to do everything that you wanted to.

One of our best decisions was to use pre-existing open-source software. Instead of having to write all sorts of ridiculous Java midi classes, we simply put jmusic.jar in the classpath, and saved ourselves a world of trouble. If you can find pre-existing software, do not hesitate to use it, especially if your language is based on something as complicated and annoying to deal with as Java midi.

Appendix Source Code

grammar.g

```
/*
 * grammar.g : the lexer and the parser in ANTLR grammar.
 *
 *  @author  Sharon  Price  -  sbp2001@columbia.edu,  Heewook  Lee  -
hl450@columbia.edu
 *
 */

class MTunesLexer extends Lexer;

options{
    k = 2;
    charVocabulary = '\3'..'\377';
    exportVocab = MTunesVocab;
}

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

protected
LOWER : 'a'..'z';

protected
```

```
UPPER : 'A'..'Z';

protected
UNDER   : '_';

protected
ALPHA   : LOWER | UPPER | UNDER ;

protected
DIGIT   : '0'..'9';

protected
PITCH options { testLiterals = false; }
      : 'C' | 'D' | 'E' | 'F' | 'G' | 'A' | 'B' | 'R';


WS      : (' ' | '\t')+
          { $setType(Token.SKIP); }
        ;

NL      : ('\n' | ('\r' '\n') => '\r' '\n' | '\r')
          { newline(); }
        ;

COMMENT : ';' (~('\n'|'\r'))* ('\n'|'\r'('\n')?)
            {$setType(Token.SKIP); newline();}
      ;

ID  options { testLiterals = true; }
        : LOWER (ALPHA|DIGIT)*
        ;

TYPE    : UPPER (LOWER)+;

LPAREN  : '(';
RPAREN  : ')';
```

```
LBRACE  : '{';
RBRACE  : '}';
COLON : ':';
COMMA   : ',';
DOT     : '.';
MULT    : '*';
PLUS    : '+';
MINUS   : '-';
DIV     : '/';
MOD     : '%';
SHARP : '#';
FLAT  : '$';
ASGN    : '=';
GE      : ">=";
LE      : "<=";
GT      : '>';
LT      : '<';
EQ      : "==";
NEQ     : "!=";
AND   : "&&";
OR    : "||";
NOT   : '!';

NUM     : (DIGIT)+ (DIV (~('0') (DIGIT)+) | (DOT (DIGIT)+))?;



NOTE  : PITCH (SHARP | FLAT)? (DIGIT)? (UNDER NUM)?;



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

INSTRUMENT
      : 'I' DIGIT (DIGIT)?
        ;

INSTRUMENT_NAME options { testLiterals = false;}
```

```
      : UPPER UPPER (UNDER | UPPER)+
      ;


class MTunesParser extends Parser;

options {
    k = 2;
    buildAST = true;
    exportVocab = MTunesVocab;
}

tokens {
      STATEMENT;
      FORLOOP_STMT;
      PRINT_STMT;
      BLOCK_STMT;
      FUNCT_CALL_EXPR;
      FOR_EXPR;
      PARAM_EXPR;
      DEC_PARAM_EXPR;
      FUNCT_STMT;
      ID_EXPR;
      MATH_TERM;
      LEFT_EXPR;
      RIGHT_EXPR;
      NUMID_EXPR;
      UMINUS;
      DEC_EXPR;
      DEC_ASGN;
      LEFT_ASGN;
      PLUS_MATH_TERM;
      MINUS_MATH_TERM;
}

{
```

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

program
      : (save_stmt)? (statement | fp_stmt)+ (play_stmt)? EOF!
      {#program = #([STATEMENT, "PROG"], program);}
      ;

fp_stmt
      : proc_def_stmt
      | funct_def_stmt
      ;

statement
      : for_stmt
      | foreach_stmt
      | if_stmt
      | assign_stmt
      | inst_assign_stmt
      | funct_call_stmt
      | add_stmt
      | print_stmt
      {#statement = #([STATEMENT, "STMT"], statement);}
      ;

for_stmt
      : "for"^ LPAREN! for_expr RPAREN! forloop_stmt
      ;
```

```
foreach_stmt
      : "foreach"^ LPAREN! ID COMMA! ID RPAREN! forloop_stmt
      ;

if_stmt
      : "if"^ LPAREN! logic_expr RPAREN! block_stmt
            (options {greedy = true;}: "else"! block_stmt)?
      ;



block_stmt
      : LBRACE! (NL!)+ (statement | break_stmt)* RBRACE! (NL!)+
      {#block_stmt = #([BLOCK_STMT, "BLOCK_STMT"], block_stmt);}
      ;

break_stmt
      : "break"^ (NL!)+
      ;

return_stmt
      : "return"^ return_expr (NL!)+
      ;

assign_stmt
      : declare_stmt (ASGN! right_expr)? (NL!)+
            { #assign_stmt = #([DEC_ASGN, "DEC_ASGN"], assign_stmt); }
      | left_expr ASGN! right_expr (NL!)+
            { #assign_stmt = #([LEFT_ASGN, "LEFT_ASGN"], assign_stmt);
}
      ;

inst_assign_stmt
      : INSTRUMENT ASGN^ INSTRUMENT_NAME (NL!)+
      ;
```

```
funct_call_stmt
      : funct_call_expr (NL!)+
      ;

funct_call_expr
      : ID LPAREN! param_expr RPAREN!
      {#funct_call_expr   =   #([FUNCT_CALL_EXPR,   "FUNCT_CALL_EXPR"],
funct_call_expr);}
      ;

add_stmt
      : "add"^ INSTRUMENT (ID | NOTE)+ (NL!)+
      ;

print_stmt
      : "print"! STRING (NL!)+
      {#print_stmt = #([PRINT_STMT, "PRINT_STMT"], print_stmt);}
      ;

proc_def_stmt
      : "procedure"^ ID LPAREN! dec_param_expr RPAREN! block_stmt
      ;

funct_def_stmt
      : "function"^ ID LPAREN! dec_param_expr   RPAREN! funct_stmt
      ;

save_stmt
      : "save"^ STRING (NL!)+
      ;

play_stmt
      : "playSong"^ (NL!)*
      ;

for_expr
```

```
      : numid_expr COMMA! numid_expr COMMA! ID
      {#for_expr = #([FOR_EXPR, "FOR_EXPR"], for_expr);}
      ;

forloop_stmt
      : LBRACE! (NL!)+ (statement)* RBRACE! (NL!)+
      {#forloop_stmt      =      #([FORLOOP_STMT,      "FORLOOP_STMT"],
forloop_stmt);}
      ;

numid_expr
      : MINUS! NUM {#numid_expr = #([UMINUS, "UMINUS"], numid_expr);}
      | NUM
      | ID
      ;

param_expr
      : (numid_expr (COMMA! numid_expr)*)?
      { #param_expr = #([PARAM_EXPR, "PARAM_EXPR"], param_expr); }
      ;

dec_param_expr
      : (declare_stmt (COMMA! declare_stmt)*)?
      {   #dec_param_expr   =   #([DEC_PARAM_EXPR,   "DEC_PARAM_EXPR"],
dec_param_expr); }
      ;

funct_stmt
      : LBRACE! (NL!)+ (statement)* (return_stmt) RBRACE! (NL!)+
      { #funct_stmt = #([FUNCT_STMT, "FUNCT_STMT"], funct_stmt); }
      ;

logic_expr
      : logic_term (OR^ logic_term)*
      ;
```

```
logic_term
      : logic_point (AND^ logic_point)*
      ;

logic_point
      : (NOT^)? relational_expr
      ;

relational_expr
      :  addsub_expr  ((GE^  |  LE^  |  GT^  |  LT^  |  EQ^  |  NEQ^)
addsub_expr)?
      ;

addsub_expr
      : multdiv_expr ((PLUS^ | MINUS^) multdiv_expr)*
      ;

multdiv_expr
      : math_term ((MULT^ | DIV^ | MOD^) math_term)*
      ;

math_term
      : PLUS! math_point
      { #math_term = #([PLUS_MATH_TERM, "PLUS_MATH_TERM"], math_term);
}
      | MINUS! math_point
      {    #math_term    =    #([MINUS_MATH_TERM,    "MINUS_MATH_TERM"],
math_term); }
      | math_point
      { #math_term = #([MATH_TERM, "MATH_TERM"], math_term); }
      ;

math_point
      : id_expr
      | funct_call_expr
      | NUM
```

```
      | LPAREN! logic_expr RPAREN!
      ;

return_expr
      : ID
      | funct_call_expr
      ;

id_expr
      : ID (DOT ("pitch" | "length" | "volume"))?
      { #id_expr = #([ID_EXPR, "ID_EXPR"], id_expr); }
      ;

declare_stmt
      : TYPE ID
      { #declare_stmt = #([DEC_EXPR, "DEC_EXPR"], declare_stmt); }
      ;

left_expr
      : id_expr
      { #left_expr = #([LEFT_EXPR, "LEFT_EXPR"], left_expr); }
      ;

right_expr
      : multdiv_expr ((PLUS^ | MINUS^) multdiv_expr)*
      | (NOTE)+
      { #right_expr = #([RIGHT_EXPR, "RIGHT_EXPR"], right_expr); }
      ;
```

walker.g

```
/*
 * walker.g
 *
 * @author Huitao Sheng - hs734@columbia.edu
 *
 */
{
      import java.io.*;
      import java.util.*;

      import jm.JMC;
      import jm.music.data.*;
      import jm.util.*;
      import jm.audio.*;
}

class MTunesWalker extends TreeParser;

options {
      importVocab = MTunesVocab;
}

{
      static Score myScore = new Score("my score");
      static Part[] partArray = new Part[16];

      MTunesInterpreter mti  = new MTunesInterpreter();
}


prog returns [MTunesInterpreter r]
{
      r = null;
}
```

```
      : #(p:STATEMENT next:.)
            {
                  MTunesVariable mtv;

                  while (#next != null) {
                        mtv = expr(#next);
                        #next = #next.getNextSibling();
                  }

                  if (mti.isSave()) {
                        mti.saveMidiFile(myScore);
                  }
                  r = mti;
            }
      ;

expr returns [MTunesVariable r]
{
      MTunesVariable a, b;
      MTunesNumber   n;
      MTunesVariable[] x;
      r = null;
}
      : #(OR a=expr right_or:.)
            {
                  if (a.getType() != MTunesVariable.BOOL) {
                        System.err.println("Inconsistent      Variable
Type: " + a.getName());
                        System.exit( 1 );
                  }
                  r    =    ((MTunesBool)a).getValue()    ?    a    :
expr(#right_or);
            }
      | #(AND a=expr right_and:.)
            {
                  if (a.getType() != MTunesVariable.BOOL) {
```

```
                        System.err.println("Inconsistent      Variable
Type: " + a.getName());
                        System.exit( 1 );
                  }
                  r = ((MTunesBool)a).getValue() ? expr(#right_and) :
a;
            }
      | #(NOT a=expr)
            {
                  if (a.getType() != MTunesVariable.BOOL) {
                        System.err.println("Inconsistent      Variable
Type: " + a.getName());
                        System.exit( 1 );
                  }
                  r = ((MTunesBool)a).not();
            }
      | #(GE a=expr b=expr)
            {
                  if (a.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + a.getName());
                        System.exit( 1 );
                  }
                  if (b.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + b.getName());
                        System.exit( 1 );
                  }
                  boolean result = ((MTunesNumber)a).getValue() >=
((MTunesNumber)b).getValue();
                  r = new MTunesBool(null, result);
            }
      | #(LE a=expr b=expr)
            {
                  if (a.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
```

```
Error: " + a.getName());
                        System.exit( 1 );
                }
                if (b.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + b.getName());
                        System.exit( 1 );
                }
                boolean result = ((MTunesNumber)a).getValue() <=
((MTunesNumber)b).getValue();
                r = new MTunesBool(null, result);
            }
      | #(GT a=expr b=expr)
            {
                if (a.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + a.getName());
                        System.exit( 1 );
                }
                if (b.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + b.getName());
                        System.exit( 1 );
                }
                boolean result = ((MTunesNumber)a).getValue() >
((MTunesNumber)b).getValue();
                r = new MTunesBool(null, result);
            }
      | #(LT a=expr b=expr)
            {
                if (a.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + a.getName());
                        System.exit( 1 );
                }
                if (b.getType() != MTunesVariable.NUM) {
```

```
                        System.err.println("Inconsistent Variable Type
Error: " + b.getName());
                        System.exit( 1 );
                  }
                  boolean result = ((MTunesNumber)a).getValue() <
((MTunesNumber)b).getValue();
                  r = new MTunesBool(null, result);
            }
      | #(EQ a=expr b=expr)
            {
                  if (a.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + a.getName());
                        System.exit( 1 );
                  }
                  if (b.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + b.getName());
                        System.exit( 1 );
                  }
                  boolean result = ((MTunesNumber)a).getValue() ==
((MTunesNumber)b).getValue();
                  r = new MTunesBool(null, result);
            }
      | #(NEQ a=expr b=expr)
            {
                  if (a.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + a.getName());
                        System.exit( 1 );
                  }
                  if (b.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + b.getName());
                        System.exit( 1 );
                  }
```

```
                  boolean  result  =  ((MTunesNumber)a).getValue()  !=
((MTunesNumber)b).getValue();
                  r = new MTunesBool(null, result);
            }
      | #(PLUS a=expr b=expr)
            {
                  Phrase p = null;
                  switch (a.getType()) {
                        case MTunesVariable.NUM:
                              if (b.getType() != MTunesVariable.NUM) {
                                    System.err.println("Inconsistent
Variable Type Error: "
                                          +  a.getName()  +  ",  "  +
b.getName());
                                    System.exit( 1 );
                              }
                              double             result             =
((MTunesNumber)a).getValue() + ((MTunesNumber)b).getValue();
                              r = new MTunesNumber(null, result);
                              break;
                        case MTunesVariable.NOTE:
                              p = new Phrase();
                              if  (b.getType()  ==  MTunesVariable.NOTE)
{

      p.add(((MTunesNote)a).getValue().copy());

      p.add(((MTunesNote)b).getValue().copy());
                              }
                              else       if       (b.getType()       ==
MTunesVariable.SEQUENCE) {

      p.add(((MTunesNote)a).getValue().copy());

      p.addNoteList(((MTunesSequence)b).getValue().copy().getNoteArray
());
```

```
                              }
                              else {
                                    System.err.println("Inconsistent
Variable Type Error: "
                                          +  a.getName()  +  ",  "  +
b.getName());
                                    System.exit( 1 );
                              }
                              r = new MTunesSequence(null, p);
                              break;
                        case MTunesVariable.SEQUENCE:
                              p = new Phrase();
                              if  (b.getType()  ==  MTunesVariable.NOTE)
{

      p.addNoteList(((MTunesSequence)a).getValue().copy().getNoteArray
());

      p.add(((MTunesNote)b).getValue().copy());
                              }
                              else       if       (b.getType()       ==
MTunesVariable.SEQUENCE) {

      p.addNoteList(((MTunesSequence)a).getValue().copy().getNoteArray
());

      p.addNoteList(((MTunesSequence)b).getValue().copy().getNoteArray
());
                              }
                              else {
                                    System.err.println("Inconsistent
Variable Type Error: "
                                          +  a.getName()  +  ",  "  +
b.getName());
                                    System.exit( 1 );
                              }
```

```
                              r = new MTunesSequence(null, p);
                              break;
                  }
            }
      | #(MINUS a=expr b=expr)
            {
                  if (a.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + a.getName());
                        System.exit( 1 );
                  }
                  if (b.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + b.getName());
                        System.exit( 1 );
                  }
                  double result = ((MTunesNumber)a).getValue() -
((MTunesNumber)b).getValue();
                  r = new MTunesNumber(null, result);
            }
      | #(MULT a=expr b=expr)
            {
                  if (a.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + a.getName());
                        System.exit( 1 );
                  }
                  if (b.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + b.getName());
                        System.exit( 1 );
                  }
                  double result = ((MTunesNumber)a).getValue() *
((MTunesNumber)b).getValue();
                  r = new MTunesNumber(null, result);
            }
```

```
      | #(DIV a=expr b=expr)
            {
                  if (a.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + a.getName());
                        System.exit( 1 );
                  }
                  if (b.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + b.getName());
                        System.exit( 1 );
                  }
                  double result = ((MTunesNumber)a).getValue() /
((MTunesNumber)b).getValue();
                  r = new MTunesNumber(null, result);
            }
      | #(MOD a=expr b=expr)
            {
                  if (a.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + a.getName());
                        System.exit( 1 );
                  }
                  if (b.getType() != MTunesVariable.NUM) {
                        System.err.println("Inconsistent Variable Type
Error: " + b.getName());
                        System.exit( 1 );
                  }
                  int i1 = (int) (((MTunesNumber)a).getValue());
                  int i2 = (int) (((MTunesNumber)b).getValue());
                  double result = i1 % i2;
                  r = new MTunesNumber(null, result);
            }
      | #(MATH_TERM a=expr) { r = a; }
      | #(PLUS_MATH_TERM a=expr) { r = a; }
      | #(MINUS_MATH_TERM a=expr)
```

```
            {
                  r          =          new          MTunesNumber(null,
((MTunesNumber)a).getValue() * (-1));
            }
      | #(ID_EXPR a=expr (expr_dot:DOT expr_id:.)*)
            {
                  if(expr_dot == null) {
                        r = a;
                  }
                  else {
                              if(expr_id.getText().compareTo("pitch")
== 0) {
                              double                d                =
((MTunesNote)a).getValue().getPitch();
                              r = new MTunesNumber(null, d);
                        }
                        else   if(expr_id.getText().compareTo("length")
== 0) {
                              double                d                =
((MTunesNote)a).getValue().getDuration();
                              r = new MTunesNumber(null, d);
                        }
                        else   if(expr_id.getText().compareTo("volume")
== 0) {
                              double                d                =
((MTunesNote)a).getValue().getDynamic();
                              r = new MTunesNumber(null, d);
                        }
                   }
            }
      | #(FUNCT_CALL_EXPR a=expr x=mexpr)
            {
                  r = mti.invokeFunct(this, a, x);
              }
      | #("return" a=expr)    { r = a; }
      | #(dec_expr:DEC_EXPR {})
```

```
            {
                  AST dec_type = dec_expr.getFirstChild();
                  AST dec_id   = dec_type.getNextSibling();
                  r      =      mti.createVariable(dec_type.getText(),
dec_id.getText());
            }
      | #(ASGN inst:INSTRUMENT inst_name:INSTRUMENT_NAME)
            {
                  int                instrumentNumber                =
Integer.parseInt(inst.getText().substring(1)) - 1;

                  if(partArray[instrumentNumber] == null) {
                        partArray[instrumentNumber]        =        new
Part(mti.getInstrumentNumber(inst_name.getText()),
                                                      instrumentNumber);
                  }
                  else {

      partArray[instrumentNumber].setInstrument(mti.getInstrumentNumbe
r(inst_name.getText()));
                  }

                  myScore.addPart(partArray[instrumentNumber]);
            }
      | #(DEC_ASGN a=expr (next:.)?)
            {
                  if (next != null) {

                        b = expr(next);

                        switch (a.getType()) {

                              case MTunesVariable.NUM:
                                    if          (b.getType()          !=
MTunesVariable.NUM) {
```

```
      System.err.println("Inconsistent Variable Type Error: "
                                                      +  a.getName()  +
", " + b.getName());
                                          System.exit( 1 );
                                    }

      ((MTunesNumber)a).setValue(((MTunesNumber)b).getValue());
                                    break;

                               case MTunesVariable.NOTE:
                                    if          (b.getType()          !=
MTunesVariable.NOTE) {

      System.err.println("Inconsistent Variable Type Error: "
                                                      +  a.getName()  +
", " + b.getName());
                                          System.exit( 1 );
                                    }

      ((MTunesNote)a).setValue(((MTunesNote)b).getValue());
                                    break;

                               case MTunesVariable.SEQUENCE:

                                    switch (b.getType()) {
                                          case MTunesVariable.NOTE:

      ((MTunesSequence)a).setValue(((MTunesNote)b).getValue());
                                                break;
                                          case
MTunesVariable.SEQUENCE:

      ((MTunesSequence)a).setValue(((MTunesSequence)b).getValue());
                                                break;
                                          default:
```

```
      System.err.println("Inconsistent Variable Type Error: "
                                                                  +
a.getName() + ", " + b.getName());
                                                  System.exit( 1 );
                                                  break;
                                        }

                                        break;
                              }

                        }

                        r = a;

                  }
            | #(LEFT_ASGN left_expr:LEFT_EXPR a=expr)
                  {
                        AST                        left                        =
left_expr.getFirstChild().getFirstChild();
                        String name = left.getText();
                        AST dot = left.getNextSibling();

                        if (a == null) {
                              /*
                               *   For   example,   right   hand   side   is   a
procedure.
                               */
                              System.err.println("Invalid          Assignment
Statement Error: " + name);
                              System.err.println("Procedure doesn't return a
value");
                              System.exit( 1 );
                        }

                        if(dot == null) {
                              mti.setVariable(name, a);
```

```
                }
                else {
                        String attr = dot.getNextSibling().getText();
                        double d = ((MTunesNumber)a).getValue();
                        mti.setNoteAttribute(name, attr, d);
                }
        }
    | #(RIGHT_EXPR aNote:NOTE
        {
                Note r1 = null;
                Phrase r2 = null;
                int singleNote = 1;

                if(aNote.getNextSibling() == null) {
                        r1 = mti.intpNote(aNote.getText());
                }
                else {
                        singleNote = 2;
                        r2 = new Phrase();
                        r2.addNote(mti.intpNote(aNote.getText()));
                }
        }
        (bNote:NOTE
                {
                        r2.addNote(mti.intpNote(bNote.getText()));
                }

        )*)
                {
                        if(singleNote == 1) {
                                r = new MTunesNote("unknown", r1);
                        }
                        else {
                                r = new MTunesSequence("unknown", r2);
                        }
                }
```

```
      | pure_note:NOTE
            {
                  Note result = mti.intpNote(aNote.getText());
                  r = new MTunesNote("unknown", result);
            }
      |   #(UMINUS   uminus:NUM)   {r   =   new   MTunesNumber(null,
Double.parseDouble(uminus.getText()) * (-1));}
      | num:NUM
            {
                  r = mti.getNumber(num.getText());
            }
      | idRef:ID
            {
                  r = mti.getVariable(idRef.getText());
                  if (r == null) {
                        System.err.println("Undefined  Symbol  Error:  "
+ idRef.getText());
                        System.exit( 1 );
                  }
            }
      | #("for" a=expr b=expr)
      | #(FOR_EXPR a=expr b=expr n=loop_counter)
            {
                  mti.forloopInit(((MTunesNumber)a).getValue(),
((MTunesNumber)b).getValue(), n);
            }
      | #(fl:FORLOOP_STMT forbody:.)
            {
                  while (mti.forloopCanProceed()) {

                        AST stmt = forbody;
                        while (stmt != null) {
                              r = expr(stmt);
                              if (mti.isReturningFromBreak()) {
                                    break;
                              }
```

```
                        stmt = stmt.getNextSibling();
                    }
                    if (mti.isReturningFromBreak()) {
                        break;
                    }
                    mti.forloopNext();

                }
                mti.completeReturnFromBreak();
                mti.forloopEnd();
            }
        | #(fe:"foreach" a=expr b=expr foreachbody:.)
            {
                mti.foreachInit((MTunesSequence)a, (MTunesNote)b);

                while (mti.foreachCanProceed()) {

                    AST stmt = foreachbody.getFirstChild();
                    while (stmt != null) {
                        r = expr(stmt);
                        if (mti.isReturningFromBreak()) {
                            break;
                        }
                        stmt = stmt.getNextSibling();
                    }
                    if (mti.isReturningFromBreak()) {
                        break;
                    }
                    mti.foreachNext();

                }
                mti.completeReturnFromBreak();
                mti.foreachEnd();
            }
        | #(PRINT_STMT text:STRING)
            {
```

```
                  System.out.println(text.getText());
            }
      | #("save" saveFile:STRING)
            {
                  mti.enableSave(saveFile.getText());
            }
      | #(BLOCK_STMT block_stmt_next:.)
            {
                  while(#block_stmt_next != null) {
                        r = expr(#block_stmt_next);
                        #block_stmt_next                              =
#block_stmt_next.getNextSibling();
                  }
            }
      | #(STATEMENT node:.)
            {
                  while (#node != null) {
                        r = expr(#node);
                        #node = #node.getNextSibling();
                  }

            }
      |     #("procedure"     procID:ID     procParamExpr:DEC_PARAM_EXPR
procBody:BLOCK_STMT)
            {
                  MTunesVariable                  mtv                 =
mti.getVariable(procID.getText());
                  if (mtv != null) {
                        System.err.println("Duplicate          Variable
Definition Error: " + procID.getText());
                        System.exit( 1 );
                  }

                  MTunesVariable[]         argList         =         new
MTunesVariable[procParamExpr.getNumberOfChildren()];
                  AST decExpr = procParamExpr.getFirstChild();
```

```
      // DEC_EXPR
                  int i = 0;
                  while (decExpr != null) {
                        AST argType = decExpr.getFirstChild();
                        AST argName = argType.getNextSibling();
                        argList[i++]                                    =
mti.createVariable(argType.getText(), argName.getText(), false);
                        decExpr = decExpr.getNextSibling();
      // next DEC_EXPR
                  }
                  MTunesFunction      mtf      =      (MTunesFunction)
mti.createVariable("Function", procID.getText());
                  mtf.setArgs(argList);
                  mtf.setBody(procBody.getFirstChild());
            }
      |     #("function"     funcID:ID     funcParamExpr:DEC_PARAM_EXPR
funcBody:FUNCT_STMT)
            {
                  MTunesVariable                mtv                 =
mti.getVariable(funcID.getText());
                  if (mtv != null) {
                        System.err.println("Duplicate          Variable
Definition Error: " + funcID.getText());
                        System.exit( 1 );
                  }

                  MTunesVariable[]        argList        =        new
MTunesVariable[funcParamExpr.getNumberOfChildren()];
                  AST decExpr = funcParamExpr.getFirstChild();
      // DEC_EXPR
                  int i = 0;
                  while (decExpr != null) {
                        AST argType = decExpr.getFirstChild();
                        AST argName = argType.getNextSibling();
                        argList[i++]                                    =
mti.createVariable(argType.getText(), argName.getText(), false);
```

```
                        decExpr = decExpr.getNextSibling();
      // next DEC_EXPR
                  }
                  MTunesFunction      mtf      =      (MTunesFunction)
mti.createVariable("Function", funcID.getText());
                  mtf.setArgs(argList);
                  mtf.setBody(funcBody.getFirstChild());
            }
      | #("playSong" {})
            {
                  if (mti.isSave()) {
                        mti.saveMidiFile(myScore);
                  }

                  mti.checkScore(myScore);

                  View.notate(myScore);
                  Play.midi(myScore);
            }
      | #("add" add_inst:INSTRUMENT add_id:ID)
            {
                  int                        instNum                        =
Integer.parseInt(add_inst.getText().substring(1)) - 1;
                  Part p = partArray[instNum];

                  if (p == null) {
                        System.err.println("Uninitialized    Instrument
Error: " + (instNum + 1));
                        System.exit( 1 );
                  }

                  do {

                        MTunesVariable              mtv              =
mti.getVariable(add_id.getText());
                        if (mtv == null) {
```

```
                              System.err.println("Undefined      Symbol
Error: " + add_id.getText());
                              System.exit( 1 );
                        }

                        switch (mtv.getType()) {
                              case MTunesVariable.NOTE:
                                    double endTime = p.getEndTime();

      p.addNote(((MTunesNote)mtv).getValue(), endTime);
                                    break;
                              case MTunesVariable.SEQUENCE:

      p.appendPhrase(((MTunesSequence)mtv).getValue());
                                    break;
                              default:
                                    System.err.println("Inconsistent
Variable Type Error: " + mtv.getName());
                                    System.exit( 1 );
                                    break;
                        }

                        add_id = add_id.getNextSibling();

                  } while (add_id != null);

            }
      | #("if" a=expr true_body:BLOCK_STMT (false_body:BLOCK_STMT)?)
            {
                  if (a.getType() != MTunesVariable.BOOL) {
                        System.err.println("Invalid Variable Type: " +
a.getName());
                        System.exit( 1 );
                  }

                  if (((MTunesBool)a).getValue()) {
```

```
                        AST stmt = true_body.getFirstChild();
                        while (stmt != null) {
                              r = expr(stmt);
                              stmt = stmt.getNextSibling();
                        }
                  }
                  else if (false_body != null) {
                        AST stmt = false_body.getFirstChild();
                        while (stmt != null) {
                              r = expr(stmt);
                              stmt = stmt.getNextSibling();
                        }
                  }

            }
      | #("break" {})
            {
                  mti.startReturnFromBreak();
            }
      ;


loop_counter returns [MTunesNumber r]
{
      r = null;
}
      : id:ID
            {
                  r = new MTunesNumber(id.getText());
            }
      ;

mexpr returns [ MTunesVariable[] rv ]
{
      MTunesVariable a;
      rv = null;
```

```
      Vector v;
}
      : #(PARAM_EXPR          { v = new Vector(); }
            ( a=expr    { v.add(a); }
            )*)         {
                              MTunesVariable[]       x       =       new
MTunesVariable[v.size()];
                              for(int i = 0; i < x.length; i++) {
                                    x[i]                                =
(MTunesVariable)(v.elementAt(i));
                              }

                              rv = x;
                        }
      ;
```

mtunesc.java

```
import java.io.*;

/**
 * Executable that performs file error checking for mTunes.
 *
 * @author Sharon Price
 *
 **/
public class mtunesc {
      public static void main(String[] args)
            throws Exception {
            // make sure arguments are good
            if (args.length != 1) {
                  System.out.println(
                        "Usage: mtunesc.exe name_of_mtunes_file.mt");
                  System.exit(1);
            }

            String mTunesScriptName = args[0];

            // make sure file exists
            if (!new File(mTunesScriptName).exists()) {
                  System.out.println(
                        "Error: "
                              + mTunesScriptName
                              + " does not exist.");
                  System.exit(1);
            }

            // make sure file is readable
            if (!new File(mTunesScriptName).canRead()) {
                  System.out.println(
                        "Error: "
                              + mTunesScriptName
```

```
                    + " is not readable.");
            System.exit(1);
      }

      // warn if not ending in .mt
      if (mTunesScriptName.length() < 3
            || !mTunesScriptName
                  .substring(
                        mTunesScriptName.length() - 3,
                        mTunesScriptName.length())
                  .equalsIgnoreCase(".mt")) {
            System.out.println(
                  "Warning: input file doesn't end in .mt");
      }

      String basename =
            beforeLastPeriod(mTunesScriptName);

      // $java mTunes <filename given in command line input>
      Process javaCreator =
            Runtime.getRuntime().exec(
                  new String[] {
                        "java",
                        "mTunes",
                        mTunesScriptName });

      copyStream(
            javaCreator.getInputStream(),
            System.out);
      copyStream(
            javaCreator.getErrorStream(),
            System.err);
      javaCreator.waitFor();

      /*
      if (!new File(basename+".java").exists()) {
```

```
                System.out.println("Errors occurred creatings java
file from script.");
                System.exit(1);
            }


            // $javac <filename but now changed to .java>
            Process javaCompiler = Runtime.getRuntime().exec(new
String[]{"javac", basename+".java"});

            copyStream(javaCompiler.getInputStream(), System.out);
            copyStream(javaCompiler.getErrorStream(), System.err);
            javaCompiler.waitFor();

            if (!new File(basename+".class").exists()) {
                System.out.println("Errors occurred compiling java
file.");
                System.exit(1);
            }

            // $java <filename without extension>
            Process javaRunner = Runtime.getRuntime().exec(new
String[]{"java", basename});

            copyStream(javaRunner.getInputStream(), System.out);
            copyStream(javaRunner.getErrorStream(), System.err);
            javaCompiler.waitFor();
            */
      }

      private static final String beforeLastPeriod(final String s) {
            int lastPeriod = s.indexOf(".");
            while (lastPeriod > -1
                  && s.indexOf(".", lastPeriod + 1) > -1)
                  lastPeriod = s.indexOf(".", lastPeriod + 1);
            return s.substring(lastPeriod + 1, s.length());
```

```
	}

	private static void copyStream(
		InputStream is,
		OutputStream os) {
		try {
			byte[] buffer = new byte[4000];

			while (true) {
				int amountRead = is.read(buffer);
				if (amountRead == -1)
					return;
				else
					os.write(buffer, 0, amountRead);
			}
		} catch (Exception e) {
			// blah
		}
	}
}
```

mTunes.java

```
import java.io.*;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.collections.AST;
import antlr.CommonAST;
import antlr.debug.misc.ASTFrame;

/*
 * Created on 2003/11/16
 */

/**
 * @author Hideki Sano
```

```
 */
public class mTunes {

      public static void main(String[] args) {

            checkArgs(args);

            try {
                  MTunesLexer ml =
                        new MTunesLexer(
                              new FileInputStream(args[0]));
                  MTunesParser mp = new MTunesParser(ml);

                  mp.program();

                  MTunesWalker mw = new MTunesWalker();
                  mw.prog(mp.getAST());
                  ASTFrame frame =
                        new ASTFrame(
                              "TREE",
                              (CommonAST) mp.getAST());
                  frame.setVisible(true);
            } catch (RecognitionException re) {
                  System.out.println(
                        "RecognitionException: " + re);
            } catch (Exception e) {
                  e.printStackTrace();
                  System.out.println(
                        "Something not mentioned above went horribly,
horribly wrong."
                              + e);
            }
      }

      private static final void checkArgs(String[] args) {
```

```
            if (args.length != 1) {
                  System.err.println(
                        "Usage: java mTunes <mTunes script file>");
                  System.exit(1);
            }

            File f = new File(args[0]);
            if (!f.canRead()) {
                  System.err.println(
                        "File Cannot Read Error: " + args[0]);
                  System.exit(1);
            }
      }
}
```

MTunesBool.java

```
/*
 * Created on 2003/12/07
 */

/**
 * @author Hideki Sano, Huitao Sheng
 */
public class MTunesBool extends MTunesVariable {

      private boolean data;

      public MTunesBool(String name, boolean data) {
            super(MTunesVariable.BOOL, name);
            this.data = data;
      }

      public MTunesBool(String name) {
            this(name, false);
      }
```

```
	public final void setValue(boolean data) {
		this.data = data;
	}

	public final boolean getValue() {
		return data;
	}

	public final MTunesBool not() {
		return new MTunesBool(null, !data);
	}

}
```

MTunesFile.java

```
/*
 * Created on 2003/12/07
 */

/**
 * @author hideki
 */
public class MTunesFile extends MTunesVariable {

	public MTunesFile(String name) {
		super(MTunesVariable.FILE, name);
	}

}
```

MTunesFunction.java

```
/*
 * Created on 2003/12/07
 */
import antlr.collections.AST;

/**
 * @author Hideki Sano, Huitao Sheng
 */
public class MTunesFunction extends MTunesVariable {

	private MTunesVariable[] args;
	private AST body;

	public MTunesFunction(
		String name,
		MTunesVariable[] args,
		AST body) {
		super(MTunesVariable.FUNCTION, name);
		this.args = args;
		this.body = body;
	}

	public MTunesFunction(String name) {
		super(MTunesVariable.FUNCTION, name);
		this.args = null;
		this.body = null;
	}

	public final void setArgs(MTunesVariable[] args) {
		this.args = args;
	}

	public final MTunesVariable[] getArgs() {
		return args;
```

```
	}

	public final void setBody(AST body) {
		this.body = body;
	}

	public final AST getBody() {
		return body;
	}
}
```

MTunesInterpreter.java

```
/*
 * Created on 2003/12/08
 */
import antlr.collections.*;
import jm.music.data.*;
import jm.constants.*;
import jm.util.*;

/**
 * @author Hideki Sano, Huitao Sheng
 */
public class MTunesInterpreter {

	private MTunesSymbolTable mtst;

	private MTunesNumber loopCounter;
	private double loopEnd;

	private boolean isSave;
	private String fileName;

	public MTunesInterpreter() {
		mtst = new MTunesSymbolTable();
		isSave = false;
	}

	public final MTunesNumber getNumber(String num) {

		String[] s = num.split("/");

		if (s.length == 1) {
			return new MTunesNumber(
				null,
				Double.parseDouble(num));
```

```
            } else {
                  return new MTunesNumber(
                        null,
                        Double.parseDouble(s[0])
                              / Double.parseDouble(s[1]));
            }

      }

      public final MTunesVariable getVariable(String name) {
            return mtst.getValue(name);
      }

      public final MTunesVariable getVariable(
            String name,
            int variableType) {
            MTunesVariable mtv = mtst.getValue(name);
            return mtv;
      }

      private final MTunesVariable createNewVariable(
            String name,
            int variableType) {

            MTunesVariable mtv = null;

            switch (variableType) {

                  case MTunesVariable.BOOL :
                        mtv = new MTunesBool(name);
                        break;

                  case MTunesVariable.FILE :
                        break;

                  case MTunesVariable.FUNCTION :
```

```
                        break;

                case MTunesVariable.NOTE :
                        break;

                case MTunesVariable.NUM :
                        mtv = new MTunesNumber(name);
                        break;

                case MTunesVariable.SEQUENCE :
                        break;

        }

        return mtv;

    }

    public final void forloopInit(
        double start,
        double end,
        MTunesNumber counter) {

        MTunesSymbolTable newTable =
                new MTunesSymbolTable();

        loopCounter = counter;
        loopCounter.setValue((int) start);
        loopEnd = (int) end;

        newTable.setValue(counter);
        newTable.setParent(mtst);

        mtst = newTable;

    }
```

```
public final boolean forloopCanProceed() {
      return loopCounter.getValue() <= loopEnd
            ? true
            : false;
}

public final void forloopNext() {
      double current = loopCounter.getValue();
      loopCounter.setValue(current + 1);
}

public final void forloopEnd() {
      mtst = mtst.getParent();
}

public final void enableSave(String fileName) {
      isSave = true;
      this.fileName = fileName;
}

private final void disableSave() {
      isSave = false;
}

public final boolean isSave() {
      return isSave;
}

public final String getSaveFileName() {
      return fileName;
}

public final void saveMidiFile(Score score) {
      Write.midi(score, fileName);
      disableSave();
```

```
	}

	private int indexNoteArray;
	private Note[] noteArrayForeach;
	private MTunesNote mtnForeach;

	public final void foreachInit(
		MTunesSequence mts,
		MTunesNote mtn) {

		MTunesSymbolTable newTable =
			new MTunesSymbolTable();
		newTable.setParent(mtst);
		mtst = newTable;

		indexNoteArray = 0;
		noteArrayForeach = mts.getValue().getNoteArray();

		mtnForeach = mtn;
		mtnForeach.setValue(
			noteArrayForeach[indexNoteArray]);

	}

	public final boolean foreachCanProceed() {
		return indexNoteArray < noteArrayForeach.length
			? true
			: false;
	}

	public final void foreachNext() {
		++indexNoteArray;
		if (foreachCanProceed()) {
			mtnForeach.setValue(
				noteArrayForeach[indexNoteArray]);
		}
```

```
	}

	public final void foreachEnd() {
		mtst = mtst.getParent();
	}

	private AST returnPoint;
	private boolean isBreaking = false;

	public final void setReturnAddressFromBreak(AST node) {
		returnPoint = node;
	}

	public final AST getReturnAddressFromBreak() {
		return returnPoint;
	}

	public final boolean isReturningFromBreak() {
		return isBreaking;
	}

	public final void startReturnFromBreak() {
		isBreaking = true;
	}

	public final void completeReturnFromBreak() {
		isBreaking = false;
	}

	public final MTunesVariable createVariable(
		String type,
		String name) {
		return createVariable(type, name, true);
	}

	public final MTunesVariable createVariable(
```

```
            String type,
            String name,
            boolean isSymbolTableRegister) {

            MTunesVariable r = null;

            if (type.compareTo("File") == 0) {
                  r = new MTunesFile(name);
            } else if (type.compareTo("Sequence") == 0) {
                  Phrase p = new Phrase();
                  r = new MTunesSequence(name, p);
            } else if (type.compareTo("Note") == 0) {
                  Note n = new Note();
                  r = new MTunesNote(name, n);
            } else if (type.compareTo("Number") == 0) {
                  r = new MTunesNumber(name);
            } else if (type.compareTo("Function") == 0) {
                  r = new MTunesFunction(name);
            }

            if (isSymbolTableRegister) {
                  mtst.setValue(r);
            }

            return r;

      }

      public final int getInstrumentNumber(String instrumentName) {

            int result = 0;

            if (instrumentName.equals("AAH")) {
                  result = ProgramChanges.AAH;
            } else if (instrumentName.equals("ABASS")) {
                  result = ProgramChanges.ABASS;
```

```
        } else if (instrumentName.equals("AC_GUITAR")) {
              result = ProgramChanges.AC_GUITAR;
        } else if (instrumentName.equals("ACCORDION")) {
              result = ProgramChanges.ACCORDION;
        } else if (
              instrumentName.equals("ACOUSTIC_BASS")) {
              result = ProgramChanges.ACOUSTIC_BASS;
        } else if (
              instrumentName.equals("ACOUSTIC_GRAND")) {
              result = ProgramChanges.ACOUSTIC_GRAND;
        } else if (
              instrumentName.equals("ACOUSTIC_GUITAR")) {
              result = ProgramChanges.ACOUSTIC_GUITAR;
        } else if (instrumentName.equals("AGOGO")) {
              result = ProgramChanges.AGOGO;
        } else if (instrumentName.equals("AHHS")) {
              result = ProgramChanges.AHHS;
        } else if (instrumentName.equals("ALTO")) {
              result = ProgramChanges.ALTO;
        } else if (instrumentName.equals("ALTO_SAX")) {
              result = ProgramChanges.ALTO_SAX;
        } else if (
              instrumentName.equals("ALTO_SAXOPHONE")) {
              result = ProgramChanges.ALTO_SAXOPHONE;
        } else if (instrumentName.equals("APPLAUSE")) {
              result = ProgramChanges.APPLAUSE;
        } else if (instrumentName.equals("ATMOSPHERE")) {
              result = ProgramChanges.ATMOSPHERE;
        } else if (instrumentName.equals("BAG_PIPES")) {
              result = ProgramChanges.BAG_PIPES;
        } else if (instrumentName.equals("BAGPIPE")) {
              result = ProgramChanges.BAGPIPE;
        } else if (instrumentName.equals("BAGPIPES")) {
              result = ProgramChanges.BAGPIPES;
        } else if (instrumentName.equals("BANDNEON")) {
              result = ProgramChanges.BANDNEON;
```

```
            } else if (instrumentName.equals("BANJO")) {
                  result = ProgramChanges.BANJO;
            } else if (instrumentName.equals("BARI")) {
                  result = ProgramChanges.BARI;
            } else if (instrumentName.equals("BARI_SAX")) {
                  result = ProgramChanges.BARI_SAX;
            } else if (instrumentName.equals("BARITONE")) {
                  result = ProgramChanges.BARITONE;
            } else if (instrumentName.equals("BARITONE_SAX")) {
                  result = ProgramChanges.BARITONE_SAX;
            } else if (
                  instrumentName.equals("BARITONE_SAXOPHONE")) {
                  result = ProgramChanges.BARITONE_SAXOPHONE;
            } else if (instrumentName.equals("BASS")) {
                  result = ProgramChanges.BASS;
            } else if (instrumentName.equals("BASSOON")) {
                  result = ProgramChanges.BASSOON;
            } else if (instrumentName.equals("BELL")) {
                  result = ProgramChanges.BELL;
            } else if (instrumentName.equals("BELLS")) {
                  result = ProgramChanges.BELLS;
            } else if (instrumentName.equals("BIRD")) {
                  result = ProgramChanges.BIRD;
            } else if (instrumentName.equals("BOTTLE")) {
                  result = ProgramChanges.BOTTLE;
            } else if (instrumentName.equals("BOTTLE_BLOW")) {
                  result = ProgramChanges.BOTTLE_BLOW;
            } else if (instrumentName.equals("BOWED_GLASS")) {
                  result = ProgramChanges.BOWED_GLASS;
            } else if (instrumentName.equals("BRASS")) {
                  result = ProgramChanges.BRASS;
            } else if (instrumentName.equals("BREATH")) {
                  result = ProgramChanges.BREATH;
            } else if (instrumentName.equals("BREATHNOISE")) {
                  result = ProgramChanges.BREATHNOISE;
            } else if (
```

```
            instrumentName.equals("BRIGHT_ACOUSTIC")) {
            result = ProgramChanges.BRIGHT_ACOUSTIC;
        } else if (instrumentName.equals("BRIGHTNESS")) {
            result = ProgramChanges.BRIGHTNESS;
        } else if (instrumentName.equals("CALLOPE")) {
            result = ProgramChanges.CALLOPE;
        } else if (instrumentName.equals("CELESTA")) {
            result = ProgramChanges.CELESTA;
        } else if (instrumentName.equals("CELESTE")) {
            result = ProgramChanges.CELESTE;
        } else if (instrumentName.equals("CELLO")) {
            result = ProgramChanges.CELLO;
        } else if (instrumentName.equals("CGUITAR")) {
            result = ProgramChanges.CGUITAR;
        } else if (instrumentName.equals("CHARANG")) {
            result = ProgramChanges.CHARANG;
        } else if (instrumentName.equals("CHIFFER")) {
            result = ProgramChanges.CHIFFER;
        } else if (instrumentName.equals("CHIFFER_LEAD")) {
            result = ProgramChanges.CHIFFER_LEAD;
        } else if (instrumentName.equals("CHOIR")) {
            result = ProgramChanges.CHOIR;
        } else if (instrumentName.equals("CHURCH_ORGAN")) {
            result = ProgramChanges.CHURCH_ORGAN;
        } else if (instrumentName.equals("CLAR")) {
            result = ProgramChanges.CLAR;
        } else if (instrumentName.equals("CLARINET")) {
            result = ProgramChanges.CLARINET;
        } else if (instrumentName.equals("CLAV")) {
            result = ProgramChanges.CLAV;
        } else if (instrumentName.equals("CLAVINET")) {
            result = ProgramChanges.CLAVINET;
        } else if (instrumentName.equals("CLEAN_GUITAR")) {
            result = ProgramChanges.CLEAN_GUITAR;
        } else if (instrumentName.equals("CONCERTINA")) {
            result = ProgramChanges.CONCERTINA;
```

```
            } else if (instrumentName.equals("CONTRA_BASS")) {
                  result = ProgramChanges.CONTRA_BASS;
            } else if (instrumentName.equals("CONTRABASS")) {
                  result = ProgramChanges.CONTRABASS;
            } else if (instrumentName.equals("CRYSTAL")) {
                  result = ProgramChanges.CRYSTAL;
            } else if (instrumentName.equals("CYMBAL")) {
                  result = ProgramChanges.CYMBAL;
            } else if (instrumentName.equals("DGUITAR")) {
                  result = ProgramChanges.DGUITAR;
            } else if (instrumentName.equals("DIST_GUITAR")) {
                  result = ProgramChanges.DIST_GUITAR;
            } else if (
                  instrumentName.equals("DISTORTED_GUITAR")) {
                  result = ProgramChanges.DISTORTED_GUITAR;
            } else if (instrumentName.equals("DOUBLE_BASS")) {
                  result = ProgramChanges.DOUBLE_BASS;
            } else if (instrumentName.equals("DROPS")) {
                  result = ProgramChanges.DROPS;
            } else if (instrumentName.equals("DRUM")) {
                  result = ProgramChanges.DRUM;
            } else if (instrumentName.equals("DX_EPIANO")) {
                  result = ProgramChanges.DX_EPIANO;
            } else if (instrumentName.equals("EBASS")) {
                  result = ProgramChanges.EBASS;
            } else if (instrumentName.equals("ECHO")) {
                  result = ProgramChanges.ECHO;
            } else if (instrumentName.equals("ECHO_DROP")) {
                  result = ProgramChanges.ECHO_DROP;
            } else if (instrumentName.equals("ECHO_DROPS")) {
                  result = ProgramChanges.ECHO_DROPS;
            } else if (instrumentName.equals("ECHOS")) {
                  result = ProgramChanges.ECHOS;
            } else if (instrumentName.equals("EL_BASS")) {
                  result = ProgramChanges.EL_BASS;
            } else if (instrumentName.equals("EL_GUITAR")) {
```

```
                result = ProgramChanges.EL_GUITAR;
            } else if (
                instrumentName.equals("ELECTRIC_BASS")) {
                result = ProgramChanges.ELECTRIC_BASS;
            } else if (
                instrumentName.equals("ELECTRIC_GRAND")) {
                result = ProgramChanges.ELECTRIC_GRAND;
            } else if (
                instrumentName.equals("ELECTRIC_GUITAR")) {
                result = ProgramChanges.ELECTRIC_GUITAR;
            } else if (
                instrumentName.equals("ELECTRIC_ORGAN")) {
                result = ProgramChanges.ELECTRIC_ORGAN;
            } else if (
                instrumentName.equals("ELECTRIC_PIANO")) {
                result = ProgramChanges.ELECTRIC_PIANO;
            } else if (instrumentName.equals("ELPIANO")) {
                result = ProgramChanges.ELPIANO;
            } else if (instrumentName.equals("ENGLISH_HORN")) {
                result = ProgramChanges.ENGLISH_HORN;
            } else if (instrumentName.equals("EPIANO")) {
                result = ProgramChanges.EPIANO;
            } else if (instrumentName.equals("EPIANO2")) {
                result = ProgramChanges.EPIANO2;
            } else if (instrumentName.equals("FANTASIA")) {
                result = ProgramChanges.FANTASIA;
            } else if (instrumentName.equals("FBASS")) {
                result = ProgramChanges.FBASS;
            } else if (instrumentName.equals("FIDDLE")) {
                result = ProgramChanges.FIDDLE;
            } else if (
                instrumentName.equals("FINGERED_BASS")) {
                result = ProgramChanges.FINGERED_BASS;
            } else if (instrumentName.equals("FLUTE")) {
                result = ProgramChanges.FLUTE;
            } else if (instrumentName.equals("FRENCH_HORN")) {
```

```
            result = ProgramChanges.FRENCH_HORN;
        } else if (instrumentName.equals("FRET")) {
            result = ProgramChanges.FRET;
        } else if (instrumentName.equals("FRET_NOISE")) {
            result = ProgramChanges.FRET_NOISE;
        } else if (instrumentName.equals("FRETLESS")) {
            result = ProgramChanges.FRETLESS;
        } else if (
            instrumentName.equals("FRETLESS_BASS")) {
            result = ProgramChanges.FRETLESS_BASS;
        } else if (instrumentName.equals("FRETNOISE")) {
            result = ProgramChanges.FRETNOISE;
        } else if (instrumentName.equals("FRETS")) {
            result = ProgramChanges.FRETS;
        } else if (instrumentName.equals("GLOCK")) {
            result = ProgramChanges.GLOCK;
        } else if (instrumentName.equals("GLOCKENSPIEL")) {
            result = ProgramChanges.GLOCKENSPIEL;
        } else if (instrumentName.equals("GMSAW_WAVE")) {
            result = ProgramChanges.GMSAW_WAVE;
        } else if (instrumentName.equals("GOBLIN")) {
            result = ProgramChanges.GOBLIN;
        } else if (instrumentName.equals("GT_HARMONICS")) {
            result = ProgramChanges.GT_HARMONICS;
        } else if (instrumentName.equals("GUITAR")) {
            result = ProgramChanges.GUITAR;
        } else if (
            instrumentName.equals("GUITAR_HARMONICS")) {
            result = ProgramChanges.GUITAR_HARMONICS;
        } else if (instrumentName.equals("HALO")) {
            result = ProgramChanges.HALO;
        } else if (instrumentName.equals("HALO_PAD")) {
            result = ProgramChanges.HALO_PAD;
        } else if (
            instrumentName.equals("HAMMOND_ORGAN")) {
            result = ProgramChanges.HAMMOND_ORGAN;
```

```
        } else if (instrumentName.equals("HARMONICA")) {
            result = ProgramChanges.HARMONICA;
        } else if (instrumentName.equals("HARMONICS")) {
            result = ProgramChanges.HARMONICS;
        } else if (instrumentName.equals("HARP")) {
            result = ProgramChanges.HARP;
        } else if (instrumentName.equals("HARPSICHORD")) {
            result = ProgramChanges.HARPSICHORD;
        } else if (instrumentName.equals("HELICOPTER")) {
            result = ProgramChanges.HELICOPTER;
        } else if (instrumentName.equals("HONKYTONK")) {
            result = ProgramChanges.HONKYTONK;
        } else if (
            instrumentName.equals("HONKYTONK_PIANO")) {
            result = ProgramChanges.HONKYTONK_PIANO;
        } else if (instrumentName.equals("HORN")) {
            result = ProgramChanges.HORN;
        } else if (instrumentName.equals("ICE_RAIN")) {
            result = ProgramChanges.ICE_RAIN;
        } else if (instrumentName.equals("ICERAIN")) {
            result = ProgramChanges.ICERAIN;
        } else if (instrumentName.equals("JAZZ_GUITAR")) {
            result = ProgramChanges.JAZZ_GUITAR;
        } else if (instrumentName.equals("JAZZ_ORGAN")) {
            result = ProgramChanges.JAZZ_ORGAN;
        } else if (instrumentName.equals("JGUITAR")) {
            result = ProgramChanges.JGUITAR;
        } else if (instrumentName.equals("KALIMBA")) {
            result = ProgramChanges.KALIMBA;
        } else if (instrumentName.equals("KOTO")) {
            result = ProgramChanges.KOTO;
        } else if (instrumentName.equals("MARIMBA")) {
            result = ProgramChanges.MARIMBA;
        } else if (instrumentName.equals("METAL_PAD")) {
            result = ProgramChanges.METAL_PAD;
        } else if (instrumentName.equals("MGUITAR")) {
```

```
            result = ProgramChanges.MGUITAR;
        } else if (instrumentName.equals("MUSIC_BOX")) {
            result = ProgramChanges.MUSIC_BOX;
        } else if (instrumentName.equals("MUTED_GUITAR")) {
            result = ProgramChanges.MUTED_GUITAR;
        } else if (
            instrumentName.equals("MUTED_TRUMPET")) {
            result = ProgramChanges.MUTED_TRUMPET;
        } else if (instrumentName.equals("NGUITAR")) {
            result = ProgramChanges.NGUITAR;
        } else if (instrumentName.equals("NYLON_GUITAR")) {
            result = ProgramChanges.NYLON_GUITAR;
        } else if (instrumentName.equals("OBOE")) {
            result = ProgramChanges.OBOE;
        } else if (instrumentName.equals("OCARINA")) {
            result = ProgramChanges.OCARINA;
        } else if (instrumentName.equals("OGUITAR")) {
            result = ProgramChanges.OGUITAR;
        } else if (instrumentName.equals("OOH")) {
            result = ProgramChanges.OOH;
        } else if (instrumentName.equals("OOHS")) {
            result = ProgramChanges.OOHS;
        } else if (
            instrumentName.equals("ORCHESTRA_HIT")) {
            result = ProgramChanges.ORCHESTRA_HIT;
        } else if (instrumentName.equals("ORGAN")) {
            result = ProgramChanges.ORGAN;
        } else if (instrumentName.equals("ORGAN2")) {
            result = ProgramChanges.ORGAN2;
        } else if (instrumentName.equals("ORGAN3")) {
            result = ProgramChanges.ORGAN3;
        } else if (
            instrumentName.equals("OVERDRIVE_GUITAR")) {
            result = ProgramChanges.OVERDRIVE_GUITAR;
        } else if (instrumentName.equals("PAD")) {
            result = ProgramChanges.PAD;
```

```
            } else if (instrumentName.equals("PAN_FLUTE")) {
                  result = ProgramChanges.PAN_FLUTE;
            } else if (instrumentName.equals("PANFLUTE")) {
                  result = ProgramChanges.PANFLUTE;
            } else if (instrumentName.equals("PBASS")) {
                  result = ProgramChanges.PBASS;
            } else if (instrumentName.equals("PHONE")) {
                  result = ProgramChanges.PHONE;
            } else if (instrumentName.equals("PIANO")) {
                  result = ProgramChanges.PIANO;
            } else if (
                  instrumentName.equals("PIANO_ACCORDION")) {
                  result = ProgramChanges.PIANO_ACCORDION;
            } else if (instrumentName.equals("PIC")) {
                  result = ProgramChanges.PIC;
            } else if (instrumentName.equals("PICC")) {
                  result = ProgramChanges.PICC;
            } else if (instrumentName.equals("PICCOLO")) {
                  result = ProgramChanges.PICCOLO;
            } else if (instrumentName.equals("PICKED_BASS")) {
                  result = ProgramChanges.PICKED_BASS;
            } else if (instrumentName.equals("PIPE_ORGAN")) {
                  result = ProgramChanges.PIPE_ORGAN;
            } else if (instrumentName.equals("PIPES")) {
                  result = ProgramChanges.PIPES;
            } else if (instrumentName.equals("PITZ")) {
                  result = ProgramChanges.PITZ;
            } else if (instrumentName.equals("PIZZ")) {
                  result = ProgramChanges.PIZZ;
            } else if (
                  instrumentName.equals("PIZZICATO_STRINGS")) {
                  result = ProgramChanges.PIZZICATO_STRINGS;
            } else if (instrumentName.equals("POLY_SYNTH")) {
                  result = ProgramChanges.POLY_SYNTH;
            } else if (instrumentName.equals("POLYSYNTH")) {
                  result = ProgramChanges.POLYSYNTH;
```

```
            } else if (instrumentName.equals("PSTRINGS")) {
                  result = ProgramChanges.PSTRINGS;
            } else if (instrumentName.equals("RAIN")) {
                  result = ProgramChanges.RAIN;
            } else if (instrumentName.equals("RECORDER")) {
                  result = ProgramChanges.RECORDER;
            } else if (instrumentName.equals("REED_ORGAN")) {
                  result = ProgramChanges.REED_ORGAN;
            } else if (
                  instrumentName.equals("REVERSE_CYMBAL")) {
                  result = ProgramChanges.REVERSE_CYMBAL;
            } else if (instrumentName.equals("RHODES")) {
                  result = ProgramChanges.RHODES;
            } else if (instrumentName.equals("SAW")) {
                  result = ProgramChanges.SAW;
            } else if (instrumentName.equals("SAWTOOTH")) {
                  result = ProgramChanges.SAWTOOTH;
            } else if (instrumentName.equals("SAX")) {
                  result = ProgramChanges.SAX;
            } else if (instrumentName.equals("SAXOPHONE")) {
                  result = ProgramChanges.SAXOPHONE;
            } else if (instrumentName.equals("SBASS")) {
                  result = ProgramChanges.SBASS;
            } else if (instrumentName.equals("SEA")) {
                  result = ProgramChanges.SEA;
            } else if (instrumentName.equals("SEASHORE")) {
                  result = ProgramChanges.SEASHORE;
            } else if (instrumentName.equals("SFX")) {
                  result = ProgramChanges.SFX;
            } else if (instrumentName.equals("SGUITAR")) {
                  result = ProgramChanges.SGUITAR;
            } else if (instrumentName.equals("SHAKUMACHI")) {
                  result = ProgramChanges.SHAKUHACHI;
            } else if (instrumentName.equals("SHAMISEN")) {
                  result = ProgramChanges.SHAMISEN;
            } else if (instrumentName.equals("SHANNAI")) {
```

```
            result = ProgramChanges.SHANNAI;
        } else if (instrumentName.equals("SITAR")) {
            result = ProgramChanges.SITAR;
        } else if (instrumentName.equals("SLAP")) {
            result = ProgramChanges.SLAP;
        } else if (instrumentName.equals("SLAP_BASS")) {
            result = ProgramChanges.SLAP_BASS;
        } else if (instrumentName.equals("SLOW_STRINGS")) {
            result = ProgramChanges.SLOW_STRINGS;
        } else if (instrumentName.equals("SOLO_VOX")) {
            result = ProgramChanges.SOLO_VOX;
        } else if (instrumentName.equals("SOP")) {
            result = ProgramChanges.SOP;
        } else if (instrumentName.equals("SOPRANO")) {
            result = ProgramChanges.SOPRANO;
        } else if (instrumentName.equals("SOPRANO_SAX")) {
            result = ProgramChanges.SOPRANO_SAX;
        } else if (
            instrumentName.equals("SOPRANO_SAXOPHONE")) {
            result = ProgramChanges.SOPRANO_SAXOPHONE;
        } else if (instrumentName.equals("SOUNDEFFECTS")) {
            result = ProgramChanges.SOUNDEFFECTS;
        } else if (instrumentName.equals("SOUNDFX")) {
            result = ProgramChanges.SOUNDFX;
        } else if (instrumentName.equals("SOUNDTRACK")) {
            result = ProgramChanges.SOUNDTRACK;
        } else if (instrumentName.equals("SPACE_VOICE")) {
            result = ProgramChanges.SPACE_VOICE;
        } else if (instrumentName.equals("SQUARE")) {
            result = ProgramChanges.SQUARE;
        } else if (instrumentName.equals("STAR_THEME")) {
            result = ProgramChanges.STAR_THEME;
        } else if (instrumentName.equals("STEEL_DRUM")) {
            result = ProgramChanges.STEEL_DRUM;
        } else if (instrumentName.equals("STEEL_DRUMS")) {
            result = ProgramChanges.STEEL_DRUMS;
```

```
        } else if (instrumentName.equals("STEEL_GUITAR")) {
              result = ProgramChanges.STEEL_GUITAR;
        } else if (instrumentName.equals("STEELDRUM")) {
              result = ProgramChanges.STEELDRUM;
        } else if (instrumentName.equals("STEELDRUMS")) {
              result = ProgramChanges.STEELDRUMS;
        } else if (instrumentName.equals("STR")) {
              result = ProgramChanges.STR;
        } else if (instrumentName.equals("STREAM")) {
              result = ProgramChanges.STREAM;
        } else if (instrumentName.equals("STRINGS")) {
              result = ProgramChanges.STRINGS;
        } else if (instrumentName.equals("SWEEP")) {
              result = ProgramChanges.SWEEP;
        } else if (instrumentName.equals("SWEEP_PAD")) {
              result = ProgramChanges.SWEEP_PAD;
        } else if (instrumentName.equals("SYN_CALLIOPE")) {
              result = ProgramChanges.SYN_CALLIOPE;
        } else if (instrumentName.equals("SYN_STRINGS")) {
              result = ProgramChanges.SYN_STRINGS;
        } else if (instrumentName.equals("SYNTH_BASS")) {
              result = ProgramChanges.SYNTH_BASS;
        } else if (instrumentName.equals("SYNTH_BRASS")) {
              result = ProgramChanges.SYNTH_BRASS;
        } else if (
              instrumentName.equals("SYNTH_CALLIOPE")) {
              result = ProgramChanges.SYNTH_CALLIOPE;
        } else if (instrumentName.equals("SYNTH_DRUM")) {
              result = ProgramChanges.SYNTH_DRUM;
        } else if (instrumentName.equals("SYNTH_DRUMS")) {
              result = ProgramChanges.SYNTH_DRUMS;
        } else if (
              instrumentName.equals("SYNTH_STRINGS")) {
              result = ProgramChanges.SYNTH_STRINGS;
        } else if (instrumentName.equals("SYNVOX")) {
              result = ProgramChanges.SYNVOX;
```

```
        } else if (instrumentName.equals("TAIKO")) {
              result = ProgramChanges.TAIKO;
        } else if (instrumentName.equals("TELEPHONE")) {
              result = ProgramChanges.TELEPHONE;
        } else if (instrumentName.equals("TENOR")) {
              result = ProgramChanges.TENOR;
        } else if (instrumentName.equals("TENOR_SAX")) {
              result = ProgramChanges.TENOR_SAX;
        } else if (
              instrumentName.equals("TENOR_SAXOPHONE")) {
              result = ProgramChanges.TENOR_SAXOPHONE;
        } else if (instrumentName.equals("THUMB_PIANO")) {
              result = ProgramChanges.THUMB_PIANO;
        } else if (instrumentName.equals("THUNDER")) {
              result = ProgramChanges.THUNDER;
        } else if (instrumentName.equals("TIMP")) {
              result = ProgramChanges.TIMP;
        } else if (instrumentName.equals("TIMPANI")) {
              result = ProgramChanges.TIMPANI;
        } else if (instrumentName.equals("TINKLE_BELL")) {
              result = ProgramChanges.TINKLE_BELL;
        } else if (instrumentName.equals("TOM")) {
              result = ProgramChanges.TOM;
        } else if (instrumentName.equals("TOM_TOM")) {
              result = ProgramChanges.TOM_TOM;
        } else if (instrumentName.equals("TOM_TOMS")) {
              result = ProgramChanges.TOM_TOMS;
        } else if (instrumentName.equals("TOMS")) {
              result = ProgramChanges.TOMS;
        } else if (instrumentName.equals("TREMOLO")) {
              result = ProgramChanges.TREMOLO;
        } else if (
              instrumentName.equals("TREMOLO_STRINGS")) {
              result = ProgramChanges.TREMOLO_STRINGS;
        } else if (instrumentName.equals("TROMBONE")) {
              result = ProgramChanges.TROMBONE;
```

```
        } else if (instrumentName.equals("TRUMPET")) {
              result = ProgramChanges.TRUMPET;
        } else if (instrumentName.equals("TUBA")) {
              result = ProgramChanges.TUBA;
        } else if (instrumentName.equals("TUBULAR_BELL")) {
              result = ProgramChanges.TUBULAR_BELL;
        } else if (
              instrumentName.equals("TUBULAR_BELLS")) {
              result = ProgramChanges.TUBULAR_BELLS;
        } else if (instrumentName.equals("VIBES")) {
              result = ProgramChanges.VIBES;
        } else if (instrumentName.equals("VIBRAPHONE")) {
              result = ProgramChanges.VIBRAPHONE;
        } else if (instrumentName.equals("VIOLA")) {
              result = ProgramChanges.VIOLA;
        } else if (instrumentName.equals("VIOLIN")) {
              result = ProgramChanges.VIOLIN;
        } else if (instrumentName.equals("VIOLIN_CELLO")) {
              result = ProgramChanges.VIOLIN_CELLO;
        } else if (instrumentName.equals("VOICE")) {
              result = ProgramChanges.VOICE;
        } else if (instrumentName.equals("VOX")) {
              result = ProgramChanges.VOX;
        } else if (instrumentName.equals("WARM_PAD")) {
              result = ProgramChanges.WARM_PAD;
        } else if (instrumentName.equals("WHISTLE")) {
              result = ProgramChanges.WHISTLE;
        } else if (instrumentName.equals("WIND")) {
              result = ProgramChanges.WIND;
        } else if (instrumentName.equals("WOODBLOCK")) {
              result = ProgramChanges.WOODBLOCK;
        } else if (instrumentName.equals("WOODBLOCKS")) {
              result = ProgramChanges.WOODBLOCKS;
        } else if (instrumentName.equals("XYLOPHONE")) {
              result = ProgramChanges.XYLOPHONE;
        } else {
```

```
                System.err.println(
                      "Invalid Instrument: " + instrumentName);
                System.exit(1);
          }

          return result;

     }

     public final MTunesVariable invokeFunct(
          MTunesWalker walker,
          MTunesVariable funct,
          MTunesVariable[] params)
          throws antlr.RecognitionException {

          MTunesSymbolTable mst = new MTunesSymbolTable();
          mst.setParent(mtst);
          mtst = mst;

          MTunesVariable[] args =
                ((MTunesFunction) funct).getArgs();
          //hsano
          for (int i = 0; i < args.length; i++) {
                MTunesVariable mtv =
                      createVariable(
                            args[i].getTypeName(),
                            args[i].getName());

                switch (mtv.getType()) {
                      case MTunesVariable.BOOL :
                            if (params[i].getType()
                                  != MTunesVariable.BOOL) {
                                  System.err.println(
                                        funct.getName()
                                              +     "     function:
Inconsist Argument");
```

```
                                    System.exit(1);
                              }
                              ((MTunesBool) mtv).setValue(
                                    ((MTunesBool) params[i])
                                          .getValue());
                              break;
                        case MTunesVariable.NUM :
                              if (params[i].getType()
                                    != MTunesVariable.NUM) {
                                    System.err.println(
                                          funct.getName()
                                                +     "     function:
Inconsist Argument");
                                    System.exit(1);
                              }
                              ((MTunesNumber) mtv).setValue(
                                    ((MTunesNumber) params[i])
                                          .getValue());
                              break;
                        case MTunesVariable.NOTE :
                              if (params[i].getType()
                                    != MTunesVariable.NOTE) {
                                    System.err.println(
                                          funct.getName()
                                                +     "     function:
Inconsist Argument");
                                    System.exit(1);
                              }
                              ((MTunesNote) mtv).setValue(
                                    ((MTunesNote) params[i])
                                          .getValue());
                              break;
                        case MTunesVariable.SEQUENCE :
                              if (params[i].getType()
                                    != MTunesVariable.SEQUENCE) {
                                    System.err.println(
```

```
                                          funct.getName()
                                                + "  function:
Inconsist Argument");
                                    System.exit(1);
                              }
                              ((MTunesSequence) mtv).setValue(
                                    ((MTunesSequence) params[i])
                                          .getValue());
                              break;
                        }

            }

            AST funct_body = ((MTunesFunction) funct).getBody();

            MTunesVariable r = null;
            while (funct_body != null) {
                  r = walker.expr(funct_body);
                  funct_body = funct_body.getNextSibling();
            }

            mtst = mst.getParent();

            return r;
      }

      public final void setVariable(
            String s,
            MTunesVariable d) {

            MTunesVariable mtv = mtst.getValue(s);

            switch (mtv.getType()) {
                  case MTunesVariable.BOOL :
                        ((MTunesBool) mtv).setValue(
                              ((MTunesBool) d).getValue());
```

```
                        break;

                case MTunesVariable.FILE :
                        break;

                case MTunesVariable.FUNCTION :
                        break;

                case MTunesVariable.NOTE :
                        ((MTunesNote) mtv).setValue(
                                ((MTunesNote) d).getValue());
                        break;

                case MTunesVariable.NUM :
                        ((MTunesNumber) mtv).setValue(
                                ((MTunesNumber) d).getValue());
                        break;

                case MTunesVariable.SEQUENCE :
                        ((MTunesSequence) mtv).setValue(
                                ((MTunesSequence) d).getValue());
                        break;
        }

        mtst.setValue(mtv);
    }

    public final void setNoteAttribute(
        String name,
        String attribute,
        double value) {

        MTunesNote aNote = (MTunesNote) mtst.getValue(name);
        Note jmNote = aNote.getValue();

        if (attribute.compareTo("pitch") == 0) {
```

```
                jmNote.setPitch((int) value);
                // Huitao: is this correct
          } else if (attribute.compareTo("length") == 0) {
                jmNote.setDuration(value);
          } else if (attribute.compareTo("volume") == 0) {
                jmNote.setDynamic((int) value);
          }

          aNote.setValue(jmNote);
          mtst.setValue(aNote);
     }

     public final Note intpNote(String note) {

          Note aNote = null;

          String[] s = note.split("_");

          int pitch = 0;
          if (s[0].startsWith("C")) {
                pitch = 0;
          } else if (s[0].startsWith("D")) {
                pitch = 2;
          } else if (s[0].startsWith("E")) {
                pitch = 4;
          } else if (s[0].startsWith("F")) {
                pitch = 5;
          } else if (s[0].startsWith("G")) {
                pitch = 7;
          } else if (s[0].startsWith("A")) {
                pitch = 9;
          } else if (s[0].startsWith("B")) {
                pitch = 11;
          } else if (s[0].startsWith("R")) {
                pitch = -2147483648;
          }
```

```
        s[0] = s[0].substring(1);

        if (s[0].indexOf('#') == 0) {
              pitch++;
              s[0] = s[0].substring(1);
        }

        if (s[0].indexOf('$') == 0) {
              pitch--;
              s[0] = s[0].substring(1);
        }

        int oct = 5;
        if (s[0].length() >= 1) {
              oct = Integer.parseInt(s[0]);
        }

        pitch += (oct + 1) * 12;

        double duration = Note.DEFAULT_RHYTHM_VALUE;
        if (s.length > 1) {
              duration = Double.parseDouble(s[1]);
        }

        if (pitch < 0) {
              pitch = -2147483648;
        }

        aNote = new Note(pitch, duration);

        return aNote;
  }

  public final void checkScore(Score s) {
```

```
            Part[] p = s.getPartArray();
            if (p.length != 0) {
                  for (int i = 0; i < p.length; i++) {
                        Phrase[] ph = p[i].getPhraseArray();
                        if (ph.length == 0) {
                              System.err.println(
                                    "Empty      Instrument      Error:
Instrument Number = "
                                          + (i + 1));
                              System.exit(1);
                        }
                  }
            }

      }

      public final void showScore(Score s) {

            System.out.println("*** show score >>>");

            Part[] p = s.getPartArray();

            for (int i = 0; i < p.length; i++) {

                  System.out.println("Part[" + i + "]");
                  System.out.println(
                        "\tChannel:      " + p[i].getChannel());
                  System.out.println(
                        "\tDenominator:  " + p[i].getDenominator());
                  System.out.println(
                        "\tEndTime:      " + p[i].getEndTime());
                  System.out.println(
                        "\tHighestPitch: "
                              + p[i].getHighestPitch());
                  System.out.println(
                        "\tInstrument:   " + p[i].getInstrument());
```

```
            Phrase[] ph = p[i].getPhraseArray();

            for (int j = 0; j < ph.length; j++) {

                  System.out.println("\t\tPhrase[" + j + "]");
                  System.out.println(
                        "\t\tBeatLength:          "
                              + ph[j].getBeatLength());
                  System.out.println(
                        "\t\tDenominator:         "
                              + ph[j].getDenominator());
                  System.out.println(
                        "\t\tEndTime:             "
                              + ph[j].getEndTime());
                  System.out.println(
                        "\t\tHighestPitch:        "
                              + ph[j].getHighestPitch());
                  System.out.println(
                        "\t\tInstrument:          "
                              + ph[j].getInstrument());
                  System.out.println(
                        "\t\tLongestRhythmValue:  "
                              + ph[j].getLongestRhythmValue());
                  System.out.println(
                        "\t\tLowestPitch:         "
                              + ph[j].getLowestPitch());
                  System.out.println(
                        "\t\tNumerator:           "
                              + ph[j].getNumerator());
                  System.out.println(
                        "\t\tPan:                 "
                              + ph[j].getPan());
                  System.out.println(
                        "\t\tShortestRhythmValue: "
                              + ph[j].getShortestRhythmValue());
```

```
                        System.out.println(
                              "\t\tSize:                "
                                    + ph[j].getSize());
                        System.out.println(
                              "\t\tStartTime:           "
                                    + ph[j].getStartTime());
                        System.out.println(
                              "\t\tTempo:               "
                                    + ph[j].getTempo());
                        System.out.println(
                              "\t\tTitle:               "
                                    + ph[j].getTitle());
                        System.out.println(
                              "\t\tVolume:              "
                                    + ph[j].getVolume());

                        Note[] no = ph[j].getNoteArray();

                        for (int k = 0; k < no.length; k++) {

                              System.out.println(
                                    "\t\t\t: Note[" + k + "]");
                              System.out.println(
                                    "\t\t\tDuration:        "
                                          + no[k].getDuration());
                              System.out.println(
                                    "\t\t\tDynamic:         "
                                          + no[k].getDynamic());
                              System.out.println(
                                    "\t\t\tFrequency:       "
                                          + no[k].getFrequency());
                              System.out.println(
                                    "\t\t\tOffset:          "
                                          + no[k].getOffset());
                              System.out.println(
                                    "\t\t\tPan:             "
```

```
                                        + no[k].getPan());
                                System.out.println(
                                      "\t\t\tPitch:           "
                                            + no[k].getPitch());
                                System.out.println(
                                      "\t\t\tRhythmValue:     "
                                            + no[k].getRhythmValue());
                                System.out.println(
                                      "\t\t\tSampleStartTime: "
                                            +
no[k].getSampleStartTime());

                           }

                     }

               }

               System.out.println("<<< show score ***");

         }

}
```

MTunesLexer.java

```
// $ANTLR 2.7.2: "grammar.g" -> "MTunesLexer.java"$

import java.io.InputStream;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.TokenStreamRecognitionException;
import antlr.CharStreamException;
import antlr.CharStreamIOException;
import antlr.ANTLRException;
import java.io.Reader;
import java.util.Hashtable;
import antlr.CharScanner;
import antlr.InputBuffer;
import antlr.ByteBuffer;
import antlr.CharBuffer;
import antlr.Token;
import antlr.CommonToken;
import antlr.RecognitionException;
import antlr.NoViableAltForCharException;
import antlr.MismatchedCharException;
import antlr.TokenStream;
import antlr.ANTLRHashString;
import antlr.LexerSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.SemanticException;

public class MTunesLexer
      extends antlr.CharScanner
      implements MTunesVocabTokenTypes, TokenStream {

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

```
	}
	public void reportError(RecognitionException e) {
		super.reportError(e);
		nr_error++;
	}
	public MTunesLexer(InputStream in) {
		this(new ByteBuffer(in));
	}
	public MTunesLexer(Reader in) {
		this(new CharBuffer(in));
	}
	public MTunesLexer(InputBuffer ib) {
		this(new LexerSharedInputState(ib));
	}
	public MTunesLexer(LexerSharedInputState state) {
		super(state);
		caseSensitiveLiterals = true;
		setCaseSensitive(true);
		literals = new Hashtable();
		literals.put(
			new ANTLRHashString("procedure", this),
			new Integer(72));
		literals.put(
			new ANTLRHashString("for", this),
			new Integer(64));
		literals.put(
			new ANTLRHashString("print", this),
			new Integer(71));
		literals.put(
			new ANTLRHashString("playSong", this),
			new Integer(75));
		literals.put(
			new ANTLRHashString("pitch", this),
			new Integer(76));
		literals.put(
			new ANTLRHashString("add", this),
```

```
                new Integer(70));
        literals.put(
                new ANTLRHashString("function", this),
                new Integer(73));
        literals.put(
                new ANTLRHashString("length", this),
                new Integer(77));
        literals.put(
                new ANTLRHashString("break", this),
                new Integer(68));
        literals.put(
                new ANTLRHashString("return", this),
                new Integer(69));
        literals.put(
                new ANTLRHashString("foreach", this),
                new Integer(65));
        literals.put(
                new ANTLRHashString("if", this),
                new Integer(66));
        literals.put(
                new ANTLRHashString("volume", this),
                new Integer(78));
        literals.put(
                new ANTLRHashString("save", this),
                new Integer(74));
        literals.put(
                new ANTLRHashString("else", this),
                new Integer(67));
    }

    public Token nextToken() throws TokenStreamException {
        Token theRetToken = null;
        tryAgain : for (;;) {
                Token _token = null;
                int _ttype = Token.INVALID_TYPE;
                resetText();
```

```
                try { // for char stream error handling
                      try { // for lexical error handling
                            switch (LA(1)) {
                                  case '\t' :
                                  case ' ' :
                                        {
                                              mWS(true);
                                              theRetToken           =
_returnToken;
                                              break;
                                        }
                                  case '\n' :
                                  case '\r' :
                                        {
                                              mNL(true);
                                              theRetToken           =
_returnToken;
                                              break;
                                        }
                                  case ';' :
                                        {
                                              mCOMMENT(true);
                                              theRetToken           =
_returnToken;
                                              break;
                                        }
                                  case 'a' :
                                  case 'b' :
                                  case 'c' :
                                  case 'd' :
                                  case 'e' :
                                  case 'f' :
                                  case 'g' :
                                  case 'h' :
                                  case 'i' :
                                  case 'j' :
```

```
                                    case 'k' :
                                    case 'l' :
                                    case 'm' :
                                    case 'n' :
                                    case 'o' :
                                    case 'p' :
                                    case 'q' :
                                    case 'r' :
                                    case 's' :
                                    case 't' :
                                    case 'u' :
                                    case 'v' :
                                    case 'w' :
                                    case 'x' :
                                    case 'y' :
                                    case 'z' :
                                          {
                                                mID(true);
                                                theRetToken           =
_returnToken;
                                                break;
                                          }
                                    case '(' :
                                          {
                                                mLPAREN(true);
                                                theRetToken           =
_returnToken;
                                                break;
                                          }
                                    case ')' :
                                          {
                                                mRPAREN(true);
                                                theRetToken           =
_returnToken;
                                                break;
                                          }
```

```
                                    case '{' :
                                          {
                                                mLBRACE(true);
                                                theRetToken           =
_returnToken;
                                                break;
                                          }
                                    case '}' :
                                          {
                                                mRBRACE(true);
                                                theRetToken           =
_returnToken;
                                                break;
                                          }
                                    case ':' :
                                          {
                                                mCOLON(true);
                                                theRetToken           =
_returnToken;
                                                break;
                                          }
                                    case ',' :
                                          {
                                                mCOMMA(true);
                                                theRetToken           =
_returnToken;
                                                break;
                                          }
                                    case '.' :
                                          {
                                                mDOT(true);
                                                theRetToken           =
_returnToken;
                                                break;
                                          }
                                    case '*' :
```

```
                                        {
                                              mMULT(true);
                                              theRetToken           =
_returnToken;
                                              break;
                                        }
                                  case '+' :
                                        {
                                              mPLUS(true);
                                              theRetToken           =
_returnToken;
                                              break;
                                        }
                                  case '-' :
                                        {
                                              mMINUS(true);
                                              theRetToken           =
_returnToken;
                                              break;
                                        }
                                  case '/' :
                                        {
                                              mDIV(true);
                                              theRetToken           =
_returnToken;
                                              break;
                                        }
                                  case '%' :
                                        {
                                              mMOD(true);
                                              theRetToken           =
_returnToken;
                                              break;
                                        }
                                  case '#' :
                                        {
```

```
                                        mSHARP(true);
                                        theRetToken           =
_returnToken;
                                        break;
                                  }
                            case '$' :
                                  {
                                        mFLAT(true);
                                        theRetToken           =
_returnToken;
                                        break;
                                  }
                            case '&' :
                                  {
                                        mAND(true);
                                        theRetToken           =
_returnToken;
                                        break;
                                  }
                            case '|' :
                                  {
                                        mOR(true);
                                        theRetToken           =
_returnToken;
                                        break;
                                  }
                            case '0' :
                            case '1' :
                            case '2' :
                            case '3' :
                            case '4' :
                            case '5' :
                            case '6' :
                            case '7' :
                            case '8' :
                            case '9' :
```

```
                                        {
                                              mNUM(true);
                                              theRetToken           =
_returnToken;
                                              break;
                                        }
                                  case '"' :
                                        {
                                              mSTRING(true);
                                              theRetToken           =
_returnToken;
                                              break;
                                        }
                                  default :
                                        if (((LA(1) >= 'A'
                                              && LA(1) <= 'Z'))
                                              && ((LA(2) >= 'a'
                                                    &&    LA(2)    <=
'z'))) {
                                              mTYPE(true);
                                              theRetToken           =
_returnToken;
                                        } else if (
                                              (LA(1) == '>')
                                                    &&    (LA(2)    ==
'=')) {
                                              mGE(true);
                                              theRetToken           =
_returnToken;
                                        } else if (
                                              (LA(1) == '<')
                                                    &&    (LA(2)    ==
'=')) {
                                              mLE(true);
                                              theRetToken           =
_returnToken;
```

```
                                        } else if (
                                              (LA(1) == '=')
                                                    &&    (LA(2)    ==
'=')) {
                                              mEQ(true);
                                              theRetToken           =
_returnToken;
                                        } else if (
                                              (LA(1) == '!')
                                                    &&    (LA(2)    ==
'=')) {
                                              mNEQ(true);
                                              theRetToken           =
_returnToken;
                                        } else if (
                                              (LA(1) == 'I')
                                                    &&   ((LA(2)   >=
'0'
                                                          &&    LA(2)
<= '9'))) {
                                              mINSTRUMENT(true);
                                              theRetToken           =
_returnToken;
                                        } else if (
                                              ((LA(1) >= 'A'
                                                    &&    LA(1)    <=
'Z'))
                                                    &&   ((LA(2)   >=
'A'
                                                          &&    LA(2)
<= 'Z'))) {

      mINSTRUMENT_NAME(true);
                                              theRetToken           =
_returnToken;
                                        } else if (
```

```
                                                (LA(1)   ==   '=')   &&
(true)) {
                                                mASGN(true);
                                                theRetToken           =
_returnToken;
                                          } else if (
                                                (LA(1)   ==   '>')   &&
(true)) {
                                                mGT(true);
                                                theRetToken           =
_returnToken;
                                          } else if (
                                                (LA(1)   ==   '<')   &&
(true)) {
                                                mLT(true);
                                                theRetToken           =
_returnToken;
                                          } else if (
                                                (LA(1)   ==   '!')   &&
(true)) {
                                                mNOT(true);
                                                theRetToken           =
_returnToken;
                                          } else if (

      (_tokenSet_0.member(LA(1)))
                                                      && (true)) {
                                                mNOTE(true);
                                                theRetToken           =
_returnToken;
                                          } else {
                                                if (LA(1) == EOF_CHAR)
{
                                                      uponEOF();
                                                      _returnToken =
                                                            makeToken(
```

```
      Token.EOF_TYPE);
                                                } else {
                                                      throw         new
NoViableAltForCharException(
                                                            (char)
LA(1),

      getFilename(),
                                                            getLine(),

      getColumn());
                                                }
                                          }
                              }
                              if (_returnToken == null)
                                    continue tryAgain;
                              // found SKIP token
                              _ttype = _returnToken.getType();
                              _ttype = testLiteralsTable(_ttype);
                              _returnToken.setType(_ttype);
                              return _returnToken;
                        } catch (RecognitionException e) {
                              throw                                    new
TokenStreamRecognitionException(e);
                        }
                  } catch (CharStreamException cse) {
                        if (cse instanceof CharStreamIOException) {
                              throw new TokenStreamIOException(
                                    ((CharStreamIOException) cse).io);
                        } else {
                              throw new TokenStreamException(
                                    cse.getMessage());
                        }
                  }
            }
```

```
	}

	protected final void mLOWER(boolean _createToken)
		throws
			RecognitionException,
			CharStreamException,
			TokenStreamException {
		int _ttype;
		Token _token = null;
		int _begin = text.length();
		_ttype = LOWER;
		int _saveIndex;

		matchRange('a', 'z');
		if (_createToken
			&& _token == null
			&& _ttype != Token.SKIP) {
			_token = makeToken(_ttype);
			_token.setText(
				new String(
					text.getBuffer(),
					_begin,
					text.length() - _begin));
		}
		_returnToken = _token;
	}

	protected final void mUPPER(boolean _createToken)
		throws
			RecognitionException,
			CharStreamException,
			TokenStreamException {
		int _ttype;
		Token _token = null;
		int _begin = text.length();
		_ttype = UPPER;
```

```
            int _saveIndex;

            matchRange('A', 'Z');
            if (_createToken
                  && _token == null
                  && _ttype != Token.SKIP) {
                  _token = makeToken(_ttype);
                  _token.setText(
                        new String(
                              text.getBuffer(),
                              _begin,
                              text.length() - _begin));
            }
            _returnToken = _token;
      }

      protected final void mUNDER(boolean _createToken)
            throws
                  RecognitionException,
                  CharStreamException,
                  TokenStreamException {
            int _ttype;
            Token _token = null;
            int _begin = text.length();
            _ttype = UNDER;
            int _saveIndex;

            match('_');
            if (_createToken
                  && _token == null
                  && _ttype != Token.SKIP) {
                  _token = makeToken(_ttype);
                  _token.setText(
                        new String(
                              text.getBuffer(),
                              _begin,
```

```
                              text.length() - _begin));
            }
            _returnToken = _token;
      }

      protected final void mALPHA(boolean _createToken)
            throws
                  RecognitionException,
                  CharStreamException,
                  TokenStreamException {
            int _ttype;
            Token _token = null;
            int _begin = text.length();
            _ttype = ALPHA;
            int _saveIndex;

            switch (LA(1)) {
                  case 'a' :
                  case 'b' :
                  case 'c' :
                  case 'd' :
                  case 'e' :
                  case 'f' :
                  case 'g' :
                  case 'h' :
                  case 'i' :
                  case 'j' :
                  case 'k' :
                  case 'l' :
                  case 'm' :
                  case 'n' :
                  case 'o' :
                  case 'p' :
                  case 'q' :
                  case 'r' :
                  case 's' :
```

```
                case 't' :
                case 'u' :
                case 'v' :
                case 'w' :
                case 'x' :
                case 'y' :
                case 'z' :
                      {
                            mLOWER(false);
                            break;
                      }
                case 'A' :
                case 'B' :
                case 'C' :
                case 'D' :
                case 'E' :
                case 'F' :
                case 'G' :
                case 'H' :
                case 'I' :
                case 'J' :
                case 'K' :
                case 'L' :
                case 'M' :
                case 'N' :
                case 'O' :
                case 'P' :
                case 'Q' :
                case 'R' :
                case 'S' :
                case 'T' :
                case 'U' :
                case 'V' :
                case 'W' :
                case 'X' :
                case 'Y' :
```

```
                case 'Z' :
                      {
                            mUPPER(false);
                            break;
                      }
                case '_' :
                      {
                            mUNDER(false);
                            break;
                      }
                default :
                      {
                            throw new NoViableAltForCharException(
                                  (char) LA(1),
                                  getFilename(),
                                  getLine(),
                                  getColumn());
                      }
          }
          if (_createToken
                && _token == null
                && _ttype != Token.SKIP) {
                _token = makeToken(_ttype);
                _token.setText(
                      new String(
                            text.getBuffer(),
                            _begin,
                            text.length() - _begin));
          }
          _returnToken = _token;
    }

    protected final void mDIGIT(boolean _createToken)
          throws
                RecognitionException,
                CharStreamException,
```

```
                TokenStreamException {
        int _ttype;
        Token _token = null;
        int _begin = text.length();
        _ttype = DIGIT;
        int _saveIndex;

        matchRange('0', '9');
        if (_createToken
              && _token == null
              && _ttype != Token.SKIP) {
              _token = makeToken(_ttype);
              _token.setText(
                    new String(
                          text.getBuffer(),
                          _begin,
                          text.length() - _begin));
        }
        _returnToken = _token;
  }

  protected final void mPITCH(boolean _createToken)
        throws
              RecognitionException,
              CharStreamException,
              TokenStreamException {
        int _ttype;
        Token _token = null;
        int _begin = text.length();
        _ttype = PITCH;
        int _saveIndex;

        switch (LA(1)) {
              case 'C' :
                    {
                          match('C');
```

```
                        break;
                }
            case 'D' :
                {
                        match('D');
                        break;
                }
            case 'E' :
                {
                        match('E');
                        break;
                }
            case 'F' :
                {
                        match('F');
                        break;
                }
            case 'G' :
                {
                        match('G');
                        break;
                }
            case 'A' :
                {
                        match('A');
                        break;
                }
            case 'B' :
                {
                        match('B');
                        break;
                }
            case 'R' :
                {
                        match('R');
                        break;
```

```
                        }
                  default :
                        {
                              throw new NoViableAltForCharException(
                                    (char) LA(1),
                                    getFilename(),
                                    getLine(),
                                    getColumn());
                        }
            }
            if (_createToken
                  && _token == null
                  && _ttype != Token.SKIP) {
                  _token = makeToken(_ttype);
                  _token.setText(
                        new String(
                              text.getBuffer(),
                              _begin,
                              text.length() - _begin));
            }
            _returnToken = _token;
      }

      public final void mWS(boolean _createToken)
            throws
                  RecognitionException,
                  CharStreamException,
                  TokenStreamException {
            int _ttype;
            Token _token = null;
            int _begin = text.length();
            _ttype = WS;
            int _saveIndex;

            {
                  int _cnt9 = 0;
```

```
                  _loop9 : do {
                        switch (LA(1)) {
                              case ' ' :
                                    {
                                          match(' ');
                                          break;
                                    }
                              case '\t' :
                                    {
                                          match('\t');
                                          break;
                                    }
                              default :
                                    {
                                          if (_cnt9 >= 1) {
                                                break _loop9;
                                          } else {
                                                throw                new
NoViableAltForCharException(
                                                      (char) LA(1),
                                                      getFilename(),
                                                      getLine(),
                                                      getColumn());
                                          }
                                    }
                        }
                        _cnt9++;
                  } while (true);
            }
            if (inputState.guessing == 0) {
                  _ttype = Token.SKIP;
            }
            if (_createToken
                  && _token == null
                  && _ttype != Token.SKIP) {
                  _token = makeToken(_ttype);
```

```
                _token.setText(
                      new String(
                            text.getBuffer(),
                            _begin,
                            text.length() - _begin));
          }
          _returnToken = _token;
    }

    public final void mNL(boolean _createToken)
          throws
                RecognitionException,
                CharStreamException,
                TokenStreamException {
          int _ttype;
          Token _token = null;
          int _begin = text.length();
          _ttype = NL;
          int _saveIndex;

          {
                boolean synPredMatched13 = false;
                if (((LA(1) == '\r') && (LA(2) == '\n'))) {
                      int _m13 = mark();
                      synPredMatched13 = true;
                      inputState.guessing++;
                      try {
                            {
                                  match('\r');
                                  match('\n');
                            }
                      } catch (RecognitionException pe) {
                            synPredMatched13 = false;
                      }
                      rewind(_m13);
                      inputState.guessing--;
```

```
                }
                if (synPredMatched13) {
                      match('\r');
                      match('\n');
                } else if ((LA(1) == '\n')) {
                      match('\n');
                } else if ((LA(1) == '\r') && (true)) {
                      match('\r');
                } else {
                      throw new NoViableAltForCharException(
                            (char) LA(1),
                            getFilename(),
                            getLine(),
                            getColumn());
                }

          }
          if (inputState.guessing == 0) {
                newline();
          }
          if (_createToken
                && _token == null
                && _ttype != Token.SKIP) {
                _token = makeToken(_ttype);
                _token.setText(
                      new String(
                            text.getBuffer(),
                            _begin,
                            text.length() - _begin));
          }
          _returnToken = _token;
    }

    public final void mCOMMENT(boolean _createToken)
          throws
                RecognitionException,
```

```
      CharStreamException,
      TokenStreamException {
int _ttype;
Token _token = null;
int _begin = text.length();
_ttype = COMMENT;
int _saveIndex;

match(';');
{
      _loop17 : do {
            if ((_tokenSet_1.member(LA(1)))) {
                  {
                        match(_tokenSet_1);
                  }
            } else {
                  break _loop17;
            }

      } while (true);
}
{
      switch (LA(1)) {
            case '\n' :
                  {
                        match('\n');
                        break;
                  }
            case '\r' :
                  {
                        match('\r');
                        {
                              if ((LA(1) == '\n')) {
                                    match('\n');
                              } else {
                              }
```

```
                                    }
                                    break;
                              }
                        default :
                              {
                                    throw                              new
NoViableAltForCharException(
                                          (char) LA(1),
                                          getFilename(),
                                          getLine(),
                                          getColumn());
                              }
                  }
            }
            if (inputState.guessing == 0) {
                  _ttype = Token.SKIP;
                  newline();
            }
            if (_createToken
                  && _token == null
                  && _ttype != Token.SKIP) {
                  _token = makeToken(_ttype);
                  _token.setText(
                        new String(
                              text.getBuffer(),
                              _begin,
                              text.length() - _begin));
            }
            _returnToken = _token;
      }

      public final void mID(boolean _createToken)
            throws
                  RecognitionException,
                  CharStreamException,
```

```
            TokenStreamException {
        int _ttype;
        Token _token = null;
        int _begin = text.length();
        _ttype = ID;
        int _saveIndex;

        mLOWER(false);
        {
            _loop22 : do {
                switch (LA(1)) {
                    case 'A' :
                    case 'B' :
                    case 'C' :
                    case 'D' :
                    case 'E' :
                    case 'F' :
                    case 'G' :
                    case 'H' :
                    case 'I' :
                    case 'J' :
                    case 'K' :
                    case 'L' :
                    case 'M' :
                    case 'N' :
                    case 'O' :
                    case 'P' :
                    case 'Q' :
                    case 'R' :
                    case 'S' :
                    case 'T' :
                    case 'U' :
                    case 'V' :
                    case 'W' :
                    case 'X' :
                    case 'Y' :
```

```
                        case 'Z' :
                        case '_' :
                        case 'a' :
                        case 'b' :
                        case 'c' :
                        case 'd' :
                        case 'e' :
                        case 'f' :
                        case 'g' :
                        case 'h' :
                        case 'i' :
                        case 'j' :
                        case 'k' :
                        case 'l' :
                        case 'm' :
                        case 'n' :
                        case 'o' :
                        case 'p' :
                        case 'q' :
                        case 'r' :
                        case 's' :
                        case 't' :
                        case 'u' :
                        case 'v' :
                        case 'w' :
                        case 'x' :
                        case 'y' :
                        case 'z' :
                              {
                                    mALPHA(false);
                                    break;
                              }
                        case '0' :
                        case '1' :
                        case '2' :
                        case '3' :
```

```
                        case '4' :
                        case '5' :
                        case '6' :
                        case '7' :
                        case '8' :
                        case '9' :
                              {
                                    mDIGIT(false);
                                    break;
                              }
                        default :
                              {
                                    break _loop22;
                              }
                  }
            } while (true);
      }
      _ttype = testLiteralsTable(_ttype);
      if (_createToken
            && _token == null
            && _ttype != Token.SKIP) {
            _token = makeToken(_ttype);
            _token.setText(
                  new String(
                        text.getBuffer(),
                        _begin,
                        text.length() - _begin));
      }
      _returnToken = _token;
}

public final void mTYPE(boolean _createToken)
      throws
            RecognitionException,
            CharStreamException,
            TokenStreamException {
```

```
            int _ttype;
            Token _token = null;
            int _begin = text.length();
            _ttype = TYPE;
            int _saveIndex;

            mUPPER(false);
            {
                  int _cnt25 = 0;
                  _loop25 : do {
                        if (((LA(1) >= 'a' && LA(1) <= 'z'))) {
                              mLOWER(false);
                        } else {
                              if (_cnt25 >= 1) {
                                    break _loop25;
                              } else {
                                    throw                              new
NoViableAltForCharException(
                                          (char) LA(1),
                                          getFilename(),
                                          getLine(),
                                          getColumn());
                              }
                        }

                        _cnt25++;
                  } while (true);
            }
            if (_createToken
                  && _token == null
                  && _ttype != Token.SKIP) {
                  _token = makeToken(_ttype);
                  _token.setText(
                        new String(
                              text.getBuffer(),
                              _begin,
```

```
                              text.length() - _begin));
            }
            _returnToken = _token;
      }

      public final void mLPAREN(boolean _createToken)
            throws
                  RecognitionException,
                  CharStreamException,
                  TokenStreamException {
            int _ttype;
            Token _token = null;
            int _begin = text.length();
            _ttype = LPAREN;
            int _saveIndex;

            match('(');
            if (_createToken
                  && _token == null
                  && _ttype != Token.SKIP) {
                  _token = makeToken(_ttype);
                  _token.setText(
                        new String(
                              text.getBuffer(),
                              _begin,
                              text.length() - _begin));
            }
            _returnToken = _token;
      }

      public final void mRPAREN(boolean _createToken)
            throws
                  RecognitionException,
                  CharStreamException,
                  TokenStreamException {
            int _ttype;
```

```
        Token _token = null;
        int _begin = text.length();
        _ttype = RPAREN;
        int _saveIndex;

        match(')');
        if (_createToken
              && _token == null
              && _ttype != Token.SKIP) {
              _token = makeToken(_ttype);
              _token.setText(
                    new String(
                          text.getBuffer(),
                          _begin,
                          text.length() - _begin));
        }
        _returnToken = _token;
  }

  public final void mLBRACE(boolean _createToken)
        throws
              RecognitionException,
              CharStreamException,
              TokenStreamException {
        int _ttype;
        Token _token = null;
        int _begin = text.length();
        _ttype = LBRACE;
        int _saveIndex;

        match('{');
        if (_createToken
              && _token == null
              && _ttype != Token.SKIP) {
              _token = makeToken(_ttype);
              _token.setText(
```

```
                        new String(
                              text.getBuffer(),
                              _begin,
                              text.length() - _begin));
            }
            _returnToken = _token;
      }

      public final void mRBRACE(boolean _createToken)
            throws
                  RecognitionException,
                  CharStreamException,
                  TokenStreamException {
            int _ttype;
            Token _token = null;
            int _begin = text.length();
            _ttype = RBRACE;
            int _saveIndex;

            match('}');
            if (_createToken
                  && _token == null
                  && _ttype != Token.SKIP) {
                  _token = makeToken(_ttype);
                  _token.setText(
                        new String(
                              text.getBuffer(),
                              _begin,
                              text.length() - _begin));
            }
            _returnToken = _token;
      }

      public final void mCOLON(boolean _createToken)
            throws
                  RecognitionException,
```

```
                CharStreamException,
                TokenStreamException {
        int _ttype;
        Token _token = null;
        int _begin = text.length();
        _ttype = COLON;
        int _saveIndex;

        match(':');
        if (_createToken
                && _token == null
                && _ttype != Token.SKIP) {
                _token = makeToken(_ttype);
                _token.setText(
                        new String(
                                text.getBuffer(),
                                _begin,
                                text.length() - _begin));
        }
        _returnToken = _token;
    }

    public final void mCOMMA(boolean _createToken)
        throws
                RecognitionException,
                CharStreamException,
                TokenStreamException {
        int _ttype;
        Token _token = null;
        int _begin = text.length();
        _ttype = COMMA;
        int _saveIndex;

        match(',');
        if (_createToken
                && _token == null
```

```
                && _ttype != Token.SKIP) {
                _token = makeToken(_ttype);
                _token.setText(
                      new String(
                            text.getBuffer(),
                            _begin,
                            text.length() - _begin));
          }
          _returnToken = _token;
     }

     public final void mDOT(boolean _createToken)
          throws
                RecognitionException,
                CharStreamException,
                TokenStreamException {
          int _ttype;
          Token _token = null;
          int _begin = text.length();
          _ttype = DOT;
          int _saveIndex;

          match('.');
          if (_createToken
                && _token == null
                && _ttype != Token.SKIP) {
                _token = makeToken(_ttype);
                _token.setText(
                      new String(
                            text.getBuffer(),
                            _begin,
                            text.length() - _begin));
          }
          _returnToken = _token;
     }
```

```
	public final void mMULT(boolean _createToken)
		throws
			RecognitionException,
			CharStreamException,
			TokenStreamException {
		int _ttype;
		Token _token = null;
		int _begin = text.length();
		_ttype = MULT;
		int _saveIndex;

		match('*');
		if (_createToken
			&& _token == null
			&& _ttype != Token.SKIP) {
			_token = makeToken(_ttype);
			_token.setText(
				new String(
					text.getBuffer(),
					_begin,
					text.length() - _begin));
		}
		_returnToken = _token;
	}

	public final void mPLUS(boolean _createToken)
		throws
			RecognitionException,
			CharStreamException,
			TokenStreamException {
		int _ttype;
		Token _token = null;
		int _begin = text.length();
		_ttype = PLUS;
		int _saveIndex;
```

```
            match('+');
            if (_createToken
                  && _token == null
                  && _ttype != Token.SKIP) {
                  _token = makeToken(_ttype);
                  _token.setText(
                        new String(
                              text.getBuffer(),
                              _begin,
                              text.length() - _begin));
            }
            _returnToken = _token;
      }

      public final void mMINUS(boolean _createToken)
            throws
                  RecognitionException,
                  CharStreamException,
                  TokenStreamException {
            int _ttype;
            Token _token = null;
            int _begin = text.length();
            _ttype = MINUS;
            int _saveIndex;

            match('-');
            if (_createToken
                  && _token == null
                  && _ttype != Token.SKIP) {
                  _token = makeToken(_ttype);
                  _token.setText(
                        new String(
                              text.getBuffer(),
                              _begin,
                              text.length() - _begin));
            }
```

```
            _returnToken = _token;
      }

      public final void mDIV(boolean _createToken)
            throws
                  RecognitionException,
                  CharStreamException,
                  TokenStreamException {
            int _ttype;
            Token _token = null;
            int _begin = text.length();
            _ttype = DIV;
            int _saveIndex;

            match('/');
            if (_createToken
                  && _token == null
                  && _ttype != Token.SKIP) {
                  _token = makeToken(_ttype);
                  _token.setText(
                        new String(
                              text.getBuffer(),
                              _begin,
                              text.length() - _begin));
            }
            _returnToken = _token;
      }

      public final void mMOD(boolean _createToken)
            throws
                  RecognitionException,
                  CharStreamException,
                  TokenStreamException {
            int _ttype;
            Token _token = null;
            int _begin = text.length();
```

```
        _ttype = MOD;
        int _saveIndex;

        match('%');
        if (_createToken
              && _token == null
              && _ttype != Token.SKIP) {
              _token = makeToken(_ttype);
              _token.setText(
                    new String(
                          text.getBuffer(),
                          _begin,
                          text.length() - _begin));
        }
        _returnToken = _token;
  }

  public final void mSHARP(boolean _createToken)
        throws
              RecognitionException,
              CharStreamException,
              TokenStreamException {
        int _ttype;
        Token _token = null;
        int _begin = text.length();
        _ttype = SHARP;
        int _saveIndex;

        match('#');
        if (_createToken
              && _token == null
              && _ttype != Token.SKIP) {
              _token = makeToken(_ttype);
              _token.setText(
                    new String(
                          text.getBuffer(),
```

```
                        _begin,
                        text.length() - _begin));
            }
            _returnToken = _token;
      }

      public final void mFLAT(boolean _createToken)
            throws
                  RecognitionException,
                  CharStreamException,
                  TokenStreamException {
            int _ttype;
            Token _token = null;
            int _begin = text.length();
            _ttype = FLAT;
            int _saveIndex;

            match('$');
            if (_createToken
                  && _token == null
                  && _ttype != Token.SKIP) {
                  _token = makeToken(_ttype);
                  _token.setText(
                        new String(
                              text.getBuffer(),
                              _begin,
                              text.length() - _begin));
            }
            _returnToken = _token;
      }

      public final void mASGN(boolean _createToken)
            throws
                  RecognitionException,
                  CharStreamException,
                  TokenStreamException {
```

```
        int _ttype;
        Token _token = null;
        int _begin = text.length();
        _ttype = ASGN;
        int _saveIndex;

        match('=');
        if (_createToken
              && _token == null
              && _ttype != Token.SKIP) {
              _token = makeToken(_ttype);
              _token.setText(
                    new String(
                          text.getBuffer(),
                          _begin,
                          text.length() - _begin));
        }
        _returnToken = _token;
  }

  public final void mGE(boolean _createToken)
        throws
              RecognitionException,
              CharStreamException,
              TokenStreamException {
        int _ttype;
        Token _token = null;
        int _begin = text.length();
        _ttype = GE;
        int _saveIndex;

        match(">=");
        if (_createToken
              && _token == null
              && _ttype != Token.SKIP) {
              _token = makeToken(_ttype);
```

```
                _token.setText(
                      new String(
                            text.getBuffer(),
                            _begin,
                            text.length() - _begin));
            }
            _returnToken = _token;
      }

      public final void mLE(boolean _createToken)
            throws
                  RecognitionException,
                  CharStreamException,
                  TokenStreamException {
            int _ttype;
            Token _token = null;
            int _begin = text.length();
            _ttype = LE;
            int _saveIndex;

            match("<=");
            if (_createToken
                  && _token == null
                  && _ttype != Token.SKIP) {
                  _token = makeToken(_ttype);
                  _token.setText(
                        new String(
                              text.getBuffer(),
                              _begin,
                              text.length() - _begin));
            }
            _returnToken = _token;
      }

      public final void mGT(boolean _createToken)
            throws
```

```
                RecognitionException,
                CharStreamException,
                TokenStreamException {
        int _ttype;
        Token _token = null;
        int _begin = text.length();
        _ttype = GT;
        int _saveIndex;

        match('>');
        if (_createToken
                && _token == null
                && _ttype != Token.SKIP) {
                _token = makeToken(_ttype);
                _token.setText(
                        new String(
                                text.getBuffer(),
                                _begin,
                                text.length() - _begin));
        }
        _returnToken = _token;
    }

    public final void mLT(boolean _createToken)
        throws
                RecognitionException,
                CharStreamException,
                TokenStreamException {
        int _ttype;
        Token _token = null;
        int _begin = text.length();
        _ttype = LT;
        int _saveIndex;

        match('<');
        if (_createToken
```

```
                && _token == null
                && _ttype != Token.SKIP) {
                _token = makeToken(_ttype);
                _token.setText(
                      new String(
                            text.getBuffer(),
                            _begin,
                            text.length() - _begin));
          }
          _returnToken = _token;
    }

    public final void mEQ(boolean _createToken)
          throws
                RecognitionException,
                CharStreamException,
                TokenStreamException {
          int _ttype;
          Token _token = null;
          int _begin = text.length();
          _ttype = EQ;
          int _saveIndex;

          match("==");
          if (_createToken
                && _token == null
                && _ttype != Token.SKIP) {
                _token = makeToken(_ttype);
                _token.setText(
                      new String(
                            text.getBuffer(),
                            _begin,
                            text.length() - _begin));
          }
          _returnToken = _token;
    }
```

```
public final void mNEQ(boolean _createToken)
      throws
            RecognitionException,
            CharStreamException,
            TokenStreamException {
      int _ttype;
      Token _token = null;
      int _begin = text.length();
      _ttype = NEQ;
      int _saveIndex;

      match("!=");
      if (_createToken
            && _token == null
            && _ttype != Token.SKIP) {
            _token = makeToken(_ttype);
            _token.setText(
                  new String(
                        text.getBuffer(),
                        _begin,
                        text.length() - _begin));
      }
      _returnToken = _token;
}

public final void mAND(boolean _createToken)
      throws
            RecognitionException,
            CharStreamException,
            TokenStreamException {
      int _ttype;
      Token _token = null;
      int _begin = text.length();
      _ttype = AND;
      int _saveIndex;
```

```
        match("&&");
        if (_createToken
              && _token == null
              && _ttype != Token.SKIP) {
              _token = makeToken(_ttype);
              _token.setText(
                    new String(
                          text.getBuffer(),
                          _begin,
                          text.length() - _begin));
        }
        _returnToken = _token;
  }

  public final void mOR(boolean _createToken)
        throws
              RecognitionException,
              CharStreamException,
              TokenStreamException {
        int _ttype;
        Token _token = null;
        int _begin = text.length();
        _ttype = OR;
        int _saveIndex;

        match("||");
        if (_createToken
              && _token == null
              && _ttype != Token.SKIP) {
              _token = makeToken(_ttype);
              _token.setText(
                    new String(
                          text.getBuffer(),
                          _begin,
                          text.length() - _begin));
```

```
        }
        _returnToken = _token;
    }

    public final void mNOT(boolean _createToken)
        throws
            RecognitionException,
            CharStreamException,
            TokenStreamException {
        int _ttype;
        Token _token = null;
        int _begin = text.length();
        _ttype = NOT;
        int _saveIndex;

        match('!');
        if (_createToken
            && _token == null
            && _ttype != Token.SKIP) {
            _token = makeToken(_ttype);
            _token.setText(
                new String(
                    text.getBuffer(),
                    _begin,
                    text.length() - _begin));
        }
        _returnToken = _token;
    }

    public final void mNUM(boolean _createToken)
        throws
            RecognitionException,
            CharStreamException,
            TokenStreamException {
        int _ttype;
        Token _token = null;
```

```
            int _begin = text.length();
            _ttype = NUM;
            int _saveIndex;

            {
                  int _cnt52 = 0;
                  _loop52 : do {
                        if (((LA(1) >= '0' && LA(1) <= '9'))) {
                              mDIGIT(false);
                        } else {
                              if (_cnt52 >= 1) {
                                    break _loop52;
                              } else {
                                    throw                              new
NoViableAltForCharException(
                                          (char) LA(1),
                                          getFilename(),
                                          getLine(),
                                          getColumn());
                              }
                        }

                        _cnt52++;
                  } while (true);
            }
            {
                  switch (LA(1)) {
                        case '/' :
                              {
                                    mDIV(false);
                                    {
                                          {
                                                match(_tokenSet_2);
                                          }
                                          {
                                                int _cnt57 = 0;
```

```
                                                _loop57 : do {
                                                      if   (((LA(1)   >=
'0'
                                                            &&    LA(1)
<= '9'))) {

      mDIGIT(false);
                                                      } else {
                                                            if  (_cnt57
>= 1) {

      break _loop57;
                                                            } else {

      throw new NoViableAltForCharException(

      (char) LA(1),

      getFilename(),

      getLine(),

      getColumn());
                                                            }
                                                      }

                                                      _cnt57++;
                                                } while (true);
                                          }
                                    }
                                    break;
                              }
                        case '.' :
                              {
                                    {
                                          mDOT(false);
```

```
                                                {
                                                      int _cnt60 = 0;
                                                      _loop60 : do {
                                                            if   (((LA(1)   >=
'0'
                                                                  &&    LA(1)
<= '9'))) {

      mDIGIT(false);
                                                            } else {
                                                                  if  (_cnt60
>= 1) {

      break _loop60;
                                                                  } else {

      throw new NoViableAltForCharException(

      (char) LA(1),

      getFilename(),

      getLine(),

      getColumn());
                                                                  }
                                                            }

                                                            _cnt60++;
                                                      } while (true);
                                                }
                                          }
                                          break;
                                    }
                              default :
                                    {
```

```
                        }
                }
        }
        if (_createToken
                && _token == null
                && _ttype != Token.SKIP) {
                _token = makeToken(_ttype);
                _token.setText(
                        new String(
                                text.getBuffer(),
                                _begin,
                                text.length() - _begin));
        }
        _returnToken = _token;
    }

    public final void mNOTE(boolean _createToken)
        throws
                RecognitionException,
                CharStreamException,
                TokenStreamException {
        int _ttype;
        Token _token = null;
        int _begin = text.length();
        _ttype = NOTE;
        int _saveIndex;

        mPITCH(false);
        {
                switch (LA(1)) {
                        case '#' :
                                {
                                        mSHARP(false);
                                        break;
                                }
                        case '$' :
```

```
                        {
                              mFLAT(false);
                              break;
                        }
                  default :
                        {
                        }
            }
      }
      {
            if (((LA(1) >= '0' && LA(1) <= '9'))) {
                  mDIGIT(false);
            } else {
            }

      }
      {
            if ((LA(1) == '_')) {
                  mUNDER(false);
                  mNUM(false);
            } else {
            }

      }
      if (_createToken
            && _token == null
            && _ttype != Token.SKIP) {
            _token = makeToken(_ttype);
            _token.setText(
                  new String(
                        text.getBuffer(),
                        _begin,
                        text.length() - _begin));
      }
      _returnToken = _token;
}
```

```
public final void mSTRING(boolean _createToken)
      throws
            RecognitionException,
            CharStreamException,
            TokenStreamException {
      int _ttype;
      Token _token = null;
      int _begin = text.length();
      _ttype = STRING;
      int _saveIndex;

      _saveIndex = text.length();
      match('"');
      text.setLength(_saveIndex);
      {
            _loop68 : do {
                  if ((_tokenSet_3.member(LA(1)))) {
                        {
                              match(_tokenSet_3);
                        }
                  } else {
                        break _loop68;
                  }

            } while (true);
      }
      _saveIndex = text.length();
      match('"');
      text.setLength(_saveIndex);
      if (_createToken
            && _token == null
            && _ttype != Token.SKIP) {
            _token = makeToken(_ttype);
            _token.setText(
                  new String(
```

```
                              text.getBuffer(),
                              _begin,
                              text.length() - _begin));
            }
            _returnToken = _token;
      }

      public final void mINSTRUMENT(boolean _createToken)
            throws
                  RecognitionException,
                  CharStreamException,
                  TokenStreamException {
            int _ttype;
            Token _token = null;
            int _begin = text.length();
            _ttype = INSTRUMENT;
            int _saveIndex;

            match('I');
            mDIGIT(false);
            {
                  if (((LA(1) >= '0' && LA(1) <= '9'))) {
                        mDIGIT(false);
                  } else {
                  }

            }
            if (_createToken
                  && _token == null
                  && _ttype != Token.SKIP) {
                  _token = makeToken(_ttype);
                  _token.setText(
                        new String(
                              text.getBuffer(),
                              _begin,
                              text.length() - _begin));
```

```
        }
        _returnToken = _token;
    }

    public final void mINSTRUMENT_NAME(boolean _createToken)
        throws
            RecognitionException,
            CharStreamException,
            TokenStreamException {
        int _ttype;
        Token _token = null;
        int _begin = text.length();
        _ttype = INSTRUMENT_NAME;
        int _saveIndex;

        mUPPER(false);
        mUPPER(false);
        {
            int _cnt73 = 0;
            _loop73 : do {
                switch (LA(1)) {
                    case '_' :
                        {
                            mUNDER(false);
                            break;
                        }
                    case 'A' :
                    case 'B' :
                    case 'C' :
                    case 'D' :
                    case 'E' :
                    case 'F' :
                    case 'G' :
                    case 'H' :
                    case 'I' :
                    case 'J' :
```

```
                              case 'K' :
                              case 'L' :
                              case 'M' :
                              case 'N' :
                              case 'O' :
                              case 'P' :
                              case 'Q' :
                              case 'R' :
                              case 'S' :
                              case 'T' :
                              case 'U' :
                              case 'V' :
                              case 'W' :
                              case 'X' :
                              case 'Y' :
                              case 'Z' :
                                    {
                                          mUPPER(false);
                                          break;
                                    }
                              default :
                                    {
                                          if (_cnt73 >= 1) {
                                                break _loop73;
                                          } else {
                                                throw               new
NoViableAltForCharException(
                                                            (char) LA(1),
                                                            getFilename(),
                                                            getLine(),
                                                            getColumn());
                                          }
                                    }
                        }
                        _cnt73++;
                  } while (true);
```

```
        }
        if (_createToken
              && _token == null
              && _ttype != Token.SKIP) {
              _token = makeToken(_ttype);
              _token.setText(
                    new String(
                          text.getBuffer(),
                          _begin,
                          text.length() - _begin));
        }
        _returnToken = _token;
  }

  private static final long[] mk_tokenSet_0() {
        long[] data = { 0L, 262398L, 0L, 0L, 0L };
        return data;
  }
  public static final BitSet _tokenSet_0 =
        new BitSet(mk_tokenSet_0());
  private static final long[] mk_tokenSet_1() {
        long[] data = new long[8];
        data[0] = -9224L;
        for (int i = 1; i <= 3; i++) {
              data[i] = -1L;
        }
        return data;
  }
  public static final BitSet _tokenSet_1 =
        new BitSet(mk_tokenSet_1());
  private static final long[] mk_tokenSet_2() {
        long[] data = new long[8];
        data[0] = -281474976710664L;
        for (int i = 1; i <= 3; i++) {
              data[i] = -1L;
        }
```

```
            return data;
      }
      public static final BitSet _tokenSet_2 =
            new BitSet(mk_tokenSet_2());
      private static final long[] mk_tokenSet_3() {
            long[] data = new long[8];
            data[0] = -17179870216L;
            for (int i = 1; i <= 3; i++) {
                  data[i] = -1L;
            }
            return data;
      }
      public static final BitSet _tokenSet_3 =
            new BitSet(mk_tokenSet_3());

}
```

MTunesNote.java

```
/*
 * Created on 2003/12/07
 */
import jm.music.data.Note;

/**
 * @author Hideki Sano, Huitao Sheng
 */
public class MTunesNote extends MTunesVariable {

	private Note note;

	public MTunesNote(String name, Note note) {
		super(MTunesVariable.NOTE, name);
		this.note = note;
	}

	public final void setValue(Note note) {
		this.note = note;
	}

	public final Note getValue() {
		return note;
	}

}
```

MTunesNumber.java

```
/*
 * Created on 2003/12/07
 */

/**
 * @author Hideki Sano, Huitao Sheng
 */
public class MTunesNumber extends MTunesVariable {

	private double data;

	public MTunesNumber(String name, double data) {
		super(MTunesVariable.NUM, name);
		this.data = data;
	}

	public MTunesNumber(String name) {
		this(name, (double) 0.0);
	}

	public final void setValue(double data) {
		this.data = data;
	}

	public final double getValue() {
		return data;
	}

}
```

MTunesParser.java

```
// $ANTLR 2.7.2: "grammar.g" -> "MTunesParser.java"$

import antlr.TokenBuffer;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.ANTLRException;
import antlr.LLkParser;
import antlr.Token;
import antlr.TokenStream;
import antlr.RecognitionException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.ParserSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.collections.AST;
import java.util.Hashtable;
import antlr.ASTFactory;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

public class MTunesParser
      extends antlr.LLkParser
      implements MTunesVocabTokenTypes {

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

```
protected MTunesParser(TokenBuffer tokenBuf, int k) {
      super(tokenBuf, k);
      tokenNames = _tokenNames;
      buildTokenTypeASTClassMap();
      astFactory =
            new ASTFactory(getTokenTypeToASTClassMap());
}

public MTunesParser(TokenBuffer tokenBuf) {
      this(tokenBuf, 2);
}

protected MTunesParser(TokenStream lexer, int k) {
      super(lexer, k);
      tokenNames = _tokenNames;
      buildTokenTypeASTClassMap();
      astFactory =
            new ASTFactory(getTokenTypeToASTClassMap());
}

public MTunesParser(TokenStream lexer) {
      this(lexer, 2);
}

public MTunesParser(ParserSharedInputState state) {
      super(state, 2);
      tokenNames = _tokenNames;
      buildTokenTypeASTClassMap();
      astFactory =
            new ASTFactory(getTokenTypeToASTClassMap());
}

public final void program()
      throws RecognitionException, TokenStreamException {
```

```
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST program_AST = null;

            try { // for error handling
                  {
                        switch (LA(1)) {
                              case LITERAL_save :
                                    {
                                          save_stmt();
                                          astFactory.addASTChild(
                                                currentAST,
                                                returnAST);
                                          break;
                                    }
                              case ID :
                              case TYPE :
                              case INSTRUMENT :
                              case LITERAL_for :
                              case LITERAL_foreach :
                              case LITERAL_if :
                              case LITERAL_add :
                              case LITERAL_print :
                              case LITERAL_procedure :
                              case LITERAL_function :
                                    {
                                          break;
                                    }
                              default :
                                    {
                                          throw                    new
NoViableAltException(
                                                LT(1),
                                                getFilename());
                                    }
                        }
```

```
                }
                {
                      int _cnt77 = 0;
                      _loop77 : do {
                            switch (LA(1)) {
                                  case ID :
                                  case TYPE :
                                  case INSTRUMENT :
                                  case LITERAL_for :
                                  case LITERAL_foreach :
                                  case LITERAL_if :
                                  case LITERAL_add :
                                  case LITERAL_print :
                                        {
                                              statement();

    astFactory.addASTChild(
                                                    currentAST,
                                                    returnAST);
                                              break;
                                        }
                                  case LITERAL_procedure :
                                  case LITERAL_function :
                                        {
                                              fp_stmt();

    astFactory.addASTChild(
                                                    currentAST,
                                                    returnAST);
                                              break;
                                        }
                                  default :
                                        {
                                              if (_cnt77 >= 1) {
                                                    break _loop77;
                                              } else {
```

```
                                                        throw         new
NoViableAltException(
                                                              LT(1),

      getFilename());
                                                  }
                                            }
                              }
                              _cnt77++;
                        } while (true);
                  }
                  {
                        switch (LA(1)) {
                              case LITERAL_playSong :
                                    {
                                          play_stmt();
                                          astFactory.addASTChild(
                                                currentAST,
                                                returnAST);
                                          break;
                                    }
                              case EOF :
                                    {
                                          break;
                                    }
                              default :
                                    {
                                          throw                    new
NoViableAltException(
                                                LT(1),
                                                getFilename());
                                    }
                        }
                  }
                  match(Token.EOF_TYPE);
                  program_AST = (AST) currentAST.root;
```

```
                program_AST =
                      (AST) astFactory.make(
                            (new ASTArray(2))
                                  .add(
                                        astFactory.create(
                                              STATEMENT,
                                              "PROG"))
                                  .add(program_AST));
                currentAST.root = program_AST;
                currentAST.child =
                      program_AST != null
                            && program_AST.getFirstChild() != null
                                  ? program_AST.getFirstChild()
                                  : program_AST;
                currentAST.advanceChildToEnd();
                program_AST = (AST) currentAST.root;
          } catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_0);
          }
          returnAST = program_AST;
    }

    public final void save_stmt()
          throws RecognitionException, TokenStreamException {

          returnAST = null;
          ASTPair currentAST = new ASTPair();
          AST save_stmt_AST = null;

          try { // for error handling
                AST tmp2_AST = null;
                tmp2_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp2_AST);
                match(LITERAL_save);
```

```
                AST tmp3_AST = null;
                tmp3_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp3_AST);
                match(STRING);
                {
                      int _cnt123 = 0;
                      _loop123 : do {
                            if ((LA(1) == NL)) {
                                  match(NL);
                            } else {
                                  if (_cnt123 >= 1) {
                                        break _loop123;
                                  } else {
                                        throw                    new
NoViableAltException(
                                              LT(1),
                                              getFilename());
                                  }
                            }

                            _cnt123++;
                      } while (true);
                }
                save_stmt_AST = (AST) currentAST.root;
          } catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_1);
          }
          returnAST = save_stmt_AST;
     }

     public final void statement()
          throws RecognitionException, TokenStreamException {

          returnAST = null;
```

```
ASTPair currentAST = new ASTPair();
AST statement_AST = null;

try { // for error handling
      switch (LA(1)) {
            case LITERAL_for :
                  {
                        for_stmt();
                        astFactory.addASTChild(
                              currentAST,
                              returnAST);
                        statement_AST =
                              (AST) currentAST.root;
                        break;
                  }
            case LITERAL_foreach :
                  {
                        foreach_stmt();
                        astFactory.addASTChild(
                              currentAST,
                              returnAST);
                        statement_AST =
                              (AST) currentAST.root;
                        break;
                  }
            case LITERAL_if :
                  {
                        if_stmt();
                        astFactory.addASTChild(
                              currentAST,
                              returnAST);
                        statement_AST =
                              (AST) currentAST.root;
                        break;
                  }
            case INSTRUMENT :
```

```
                        {
                              inst_assign_stmt();
                              astFactory.addASTChild(
                                    currentAST,
                                    returnAST);
                              statement_AST =
                                    (AST) currentAST.root;
                              break;
                        }
                  case LITERAL_add :
                        {
                              add_stmt();
                              astFactory.addASTChild(
                                    currentAST,
                                    returnAST);
                              statement_AST =
                                    (AST) currentAST.root;
                              break;
                        }
                  case LITERAL_print :
                        {
                              print_stmt();
                              astFactory.addASTChild(
                                    currentAST,
                                    returnAST);
                              statement_AST =
                                    (AST) currentAST.root;
                              statement_AST =
                                    (AST) astFactory.make(
                                          (new ASTArray(2))
                                                .add(

astFactory.create(

STATEMENT,
```

```
      "STMT"))

      .add(statement_AST));
                                    currentAST.root = statement_AST;
                                    currentAST.child =
                                          statement_AST != null
                                                && statement_AST
                                                      .getFirstChild()
                                                      != null
                                                      ? statement_AST

      .getFirstChild()
                                                      : statement_AST;
                                    currentAST.advanceChildToEnd();
                                    statement_AST =
                                          (AST) currentAST.root;
                                    break;
                              }
                        default :
                              if ((LA(1) == ID || LA(1) == TYPE)
                                    && (LA(2) == ID
                                          || LA(2) == DOT
                                          || LA(2) == ASGN)) {
                                    assign_stmt();
                                    astFactory.addASTChild(
                                          currentAST,
                                          returnAST);
                                    statement_AST =
                                          (AST) currentAST.root;
                              } else if (
                                    (LA(1) == ID)
                                          && (LA(2) == LPAREN)) {
                                    funct_call_stmt();
                                    astFactory.addASTChild(
                                          currentAST,
                                          returnAST);
```

```
                                    statement_AST =
                                          (AST) currentAST.root;
                              } else {
                                    throw new NoViableAltException(
                                          LT(1),
                                          getFilename());
                              }
                        }
                  } catch (RecognitionException ex) {
                        reportError(ex);
                        consume();
                        consumeUntil(_tokenSet_2);
                  }
                  returnAST = statement_AST;
            }

            public final void fp_stmt()
                  throws RecognitionException, TokenStreamException {

                  returnAST = null;
                  ASTPair currentAST = new ASTPair();
                  AST fp_stmt_AST = null;

                  try { // for error handling
                        switch (LA(1)) {
                              case LITERAL_procedure :
                                    {
                                          proc_def_stmt();
                                          astFactory.addASTChild(
                                                currentAST,
                                                returnAST);
                                          fp_stmt_AST         =         (AST)
currentAST.root;
                                          break;
                                    }
                              case LITERAL_function :
```

```
                              {
                                    funct_def_stmt();
                                    astFactory.addASTChild(
                                          currentAST,
                                          returnAST);
                                    fp_stmt_AST         =         (AST)
currentAST.root;
                                    break;
                              }
                        default :
                              {
                                    throw new NoViableAltException(
                                          LT(1),
                                          getFilename());
                              }
                  }
            } catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_3);
            }
            returnAST = fp_stmt_AST;
      }

      public final void play_stmt()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST play_stmt_AST = null;

            try { // for error handling
                  AST tmp5_AST = null;
                  tmp5_AST = astFactory.create(LT(1));
                  astFactory.makeASTRoot(currentAST, tmp5_AST);
                  match(LITERAL_playSong);
```

```
                {
                      _loop126 : do {
                            if ((LA(1) == NL)) {
                                  match(NL);
                            } else {
                                  break _loop126;
                            }

                      } while (true);
                }
                play_stmt_AST = (AST) currentAST.root;
          } catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_0);
          }
          returnAST = play_stmt_AST;
    }

    public final void proc_def_stmt()
          throws RecognitionException, TokenStreamException {

          returnAST = null;
          ASTPair currentAST = new ASTPair();
          AST proc_def_stmt_AST = null;

          try { // for error handling
                AST tmp7_AST = null;
                tmp7_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp7_AST);
                match(LITERAL_procedure);
                AST tmp8_AST = null;
                tmp8_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp8_AST);
                match(ID);
                match(LPAREN);
```

```
                dec_param_expr();
                astFactory.addASTChild(currentAST, returnAST);
                match(RPAREN);
                block_stmt();
                astFactory.addASTChild(currentAST, returnAST);
                proc_def_stmt_AST = (AST) currentAST.root;
          } catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_3);
          }
          returnAST = proc_def_stmt_AST;
    }

    public final void funct_def_stmt()
          throws RecognitionException, TokenStreamException {

          returnAST = null;
          ASTPair currentAST = new ASTPair();
          AST funct_def_stmt_AST = null;

          try { // for error handling
                AST tmp11_AST = null;
                tmp11_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp11_AST);
                match(LITERAL_function);
                AST tmp12_AST = null;
                tmp12_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp12_AST);
                match(ID);
                match(LPAREN);
                dec_param_expr();
                astFactory.addASTChild(currentAST, returnAST);
                match(RPAREN);
                funct_stmt();
                astFactory.addASTChild(currentAST, returnAST);
```

```
                funct_def_stmt_AST = (AST) currentAST.root;
          } catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_3);
          }
          returnAST = funct_def_stmt_AST;
    }

    public final void for_stmt()
          throws RecognitionException, TokenStreamException {

          returnAST = null;
          ASTPair currentAST = new ASTPair();
          AST for_stmt_AST = null;

          try { // for error handling
                AST tmp15_AST = null;
                tmp15_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp15_AST);
                match(LITERAL_for);
                match(LPAREN);
                for_expr();
                astFactory.addASTChild(currentAST, returnAST);
                match(RPAREN);
                forloop_stmt();
                astFactory.addASTChild(currentAST, returnAST);
                for_stmt_AST = (AST) currentAST.root;
          } catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_2);
          }
          returnAST = for_stmt_AST;
    }
```

```
	public final void foreach_stmt()
		throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST foreach_stmt_AST = null;

		try { // for error handling
			AST tmp18_AST = null;
			tmp18_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp18_AST);
			match(LITERAL_foreach);
			match(LPAREN);
			AST tmp20_AST = null;
			tmp20_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp20_AST);
			match(ID);
			match(COMMA);
			AST tmp22_AST = null;
			tmp22_AST = astFactory.create(LT(1));
			astFactory.addASTChild(currentAST, tmp22_AST);
			match(ID);
			match(RPAREN);
			forloop_stmt();
			astFactory.addASTChild(currentAST, returnAST);
			foreach_stmt_AST = (AST) currentAST.root;
		} catch (RecognitionException ex) {
			reportError(ex);
			consume();
			consumeUntil(_tokenSet_2);
		}
		returnAST = foreach_stmt_AST;
	}

	public final void if_stmt()
		throws RecognitionException, TokenStreamException {
```

```
returnAST = null;
ASTPair currentAST = new ASTPair();
AST if_stmt_AST = null;

try { // for error handling
      AST tmp24_AST = null;
      tmp24_AST = astFactory.create(LT(1));
      astFactory.makeASTRoot(currentAST, tmp24_AST);
      match(LITERAL_if);
      match(LPAREN);
      logic_expr();
      astFactory.addASTChild(currentAST, returnAST);
      match(RPAREN);
      block_stmt();
      astFactory.addASTChild(currentAST, returnAST);
      {
            switch (LA(1)) {
                  case LITERAL_else :
                        {
                              match(LITERAL_else);
                              block_stmt();
                              astFactory.addASTChild(
                                    currentAST,
                                    returnAST);
                              break;
                        }
                  case EOF :
                  case ID :
                  case TYPE :
                  case RBRACE :
                  case INSTRUMENT :
                  case LITERAL_for :
                  case LITERAL_foreach :
                  case LITERAL_if :
                  case LITERAL_break :
```

```
                              case LITERAL_return :
                              case LITERAL_add :
                              case LITERAL_print :
                              case LITERAL_procedure :
                              case LITERAL_function :
                              case LITERAL_playSong :
                                    {
                                          break;
                                    }
                              default :
                                    {
                                          throw                    new
NoViableAltException(
                                                LT(1),
                                                getFilename());
                                    }
                        }
                  }
                  if_stmt_AST = (AST) currentAST.root;
            } catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_2);
            }
            returnAST = if_stmt_AST;
      }

      public final void assign_stmt()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST assign_stmt_AST = null;

            try { // for error handling
                  switch (LA(1)) {
```

```
                        case TYPE :
                              {
                                    declare_stmt();
                                    astFactory.addASTChild(
                                          currentAST,
                                          returnAST);
                                    {
                                          switch (LA(1)) {
                                                case ASGN :
                                                      {

      match(ASGN);

      right_expr();
                                                                  astFactory

      .addASTChild(

      currentAST,

      returnAST);
                                                                  break;
                                                      }
                                                case NL :
                                                      {
                                                                  break;
                                                      }
                                                default :
                                                      {
                                                                  throw   new
NoViableAltException(

      LT(1),

      getFilename());
                                                      }
```

```
                                        }
                                  }
                                  {
                                        int _cnt101 = 0;
                                        _loop101 : do {
                                              if ((LA(1) == NL)) {
                                                    match(NL);
                                              } else {
                                                    if   (_cnt101   >=
1) {
                                                          break
_loop101;
                                                    } else {
                                                          throw   new
NoViableAltException(
      LT(1),
      getFilename());
                                                    }
                                              }

                                              _cnt101++;
                                        } while (true);
                                  }
                                  assign_stmt_AST =
                                        (AST) currentAST.root;
                                  assign_stmt_AST =
                                        (AST) astFactory.make(
                                              (new ASTArray(2))
                                                    .add(
      astFactory.create(
      DEC_ASGN,
```

```
      "DEC_ASGN"))

      .add(assign_stmt_AST));
                                    currentAST.root = assign_stmt_AST;
                                    currentAST.child =
                                          assign_stmt_AST != null
                                                && assign_stmt_AST
                                                      .getFirstChild()
                                                      != null
                                                      ?
assign_stmt_AST

      .getFirstChild()
                                                      :
assign_stmt_AST;
                                    currentAST.advanceChildToEnd();
                                    assign_stmt_AST =
                                          (AST) currentAST.root;
                                    break;
                              }
                        case ID :
                              {
                                    left_expr();
                                    astFactory.addASTChild(
                                          currentAST,
                                          returnAST);
                                    match(ASGN);
                                    right_expr();
                                    astFactory.addASTChild(
                                          currentAST,
                                          returnAST);
                                    {
                                          int _cnt103 = 0;
                                          _loop103 : do {
                                                if ((LA(1) == NL)) {
                                                      match(NL);
```

```
                                                } else {
                                                      if   (_cnt103   >=
1) {
                                                            break
_loop103;
                                                      } else {
                                                            throw   new
NoViableAltException(
      LT(1),
      getFilename());
                                                      }
                                                }

                                                _cnt103++;
                                          } while (true);
                                    }
                                    assign_stmt_AST =
                                          (AST) currentAST.root;
                                    assign_stmt_AST =
                                          (AST) astFactory.make(
                                                (new ASTArray(2))
                                                      .add(
      astFactory.create(
      LEFT_ASGN,
      "LEFT_ASGN"))
      .add(assign_stmt_AST));
                                    currentAST.root = assign_stmt_AST;
                                    currentAST.child =
                                          assign_stmt_AST != null
                                                && assign_stmt_AST
```

```
                                                      .getFirstChild()
                                                      != null
                                                      ?
assign_stmt_AST

      .getFirstChild()
                                                      :
assign_stmt_AST;
                                    currentAST.advanceChildToEnd();
                                    assign_stmt_AST =
                                          (AST) currentAST.root;
                                    break;
                              }
                        default :
                              {
                                    throw new NoViableAltException(
                                          LT(1),
                                          getFilename());
                              }
                  }
            } catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_2);
            }
            returnAST = assign_stmt_AST;
      }

      public final void inst_assign_stmt()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST inst_assign_stmt_AST = null;

            try { // for error handling
```

```
                AST tmp32_AST = null;
                tmp32_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp32_AST);
                match(INSTRUMENT);
                AST tmp33_AST = null;
                tmp33_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp33_AST);
                match(ASGN);
                AST tmp34_AST = null;
                tmp34_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp34_AST);
                match(INSTRUMENT_NAME);
                {
                      int _cnt106 = 0;
                      _loop106 : do {
                            if ((LA(1) == NL)) {
                                  match(NL);
                            } else {
                                  if (_cnt106 >= 1) {
                                        break _loop106;
                                  } else {
                                        throw                    new
NoViableAltException(
                                              LT(1),
                                              getFilename());
                                  }
                            }

                            _cnt106++;
                      } while (true);
                }
                inst_assign_stmt_AST = (AST) currentAST.root;
          } catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_2);
```

```
            }
            returnAST = inst_assign_stmt_AST;
      }

      public final void funct_call_stmt()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST funct_call_stmt_AST = null;

            try { // for error handling
                  funct_call_expr();
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                        int _cnt109 = 0;
                        _loop109 : do {
                              if ((LA(1) == NL)) {
                                    match(NL);
                              } else {
                                    if (_cnt109 >= 1) {
                                          break _loop109;
                                    } else {
                                          throw                    new
NoViableAltException(
                                                LT(1),
                                                getFilename());
                                    }
                              }

                              _cnt109++;
                        } while (true);
                  }
                  funct_call_stmt_AST = (AST) currentAST.root;
            } catch (RecognitionException ex) {
                  reportError(ex);
```

```
                consume();
                consumeUntil(_tokenSet_2);
          }
          returnAST = funct_call_stmt_AST;
    }

    public final void add_stmt()
          throws RecognitionException, TokenStreamException {

          returnAST = null;
          ASTPair currentAST = new ASTPair();
          AST add_stmt_AST = null;

          try { // for error handling
                AST tmp37_AST = null;
                tmp37_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp37_AST);
                match(LITERAL_add);
                AST tmp38_AST = null;
                tmp38_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp38_AST);
                match(INSTRUMENT);
                {
                      int _cnt113 = 0;
                      _loop113 : do {
                            switch (LA(1)) {
                                  case ID :
                                        {
                                              AST tmp39_AST = null;
                                              tmp39_AST =

    astFactory.create(
                                                                  LT(1));

    astFactory.addASTChild(
                                                        currentAST,
```

```
                                                      tmp39_AST);
                                                match(ID);
                                                break;
                                          }
                                    case NOTE :
                                          {
                                                AST tmp40_AST = null;
                                                tmp40_AST =

      astFactory.create(
                                                            LT(1));

      astFactory.addASTChild(
                                                      currentAST,
                                                      tmp40_AST);
                                                match(NOTE);
                                                break;
                                          }
                                    default :
                                          {
                                                if (_cnt113 >= 1) {
                                                      break _loop113;
                                                } else {
                                                      throw         new
NoViableAltException(
                                                            LT(1),

      getFilename());
                                                }
                                          }
                              }
                              _cnt113++;
                        } while (true);
                  }
                  {
                        int _cnt115 = 0;
```

```
                        _loop115 : do {
                              if ((LA(1) == NL)) {
                                    match(NL);
                              } else {
                                    if (_cnt115 >= 1) {
                                          break _loop115;
                                    } else {
                                          throw                    new
NoViableAltException(
                                                LT(1),
                                                getFilename());
                                    }
                              }

                              _cnt115++;
                        } while (true);
                  }
                  add_stmt_AST = (AST) currentAST.root;
            } catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_2);
            }
            returnAST = add_stmt_AST;
      }

      public final void print_stmt()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST print_stmt_AST = null;

            try { // for error handling
                  match(LITERAL_print);
                  AST tmp43_AST = null;
```

```
                  tmp43_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp43_AST);
                  match(STRING);
                  {
                        int _cnt118 = 0;
                        _loop118 : do {
                              if ((LA(1) == NL)) {
                                    match(NL);
                              } else {
                                    if (_cnt118 >= 1) {
                                          break _loop118;
                                    } else {
                                          throw                    new
NoViableAltException(
                                                LT(1),
                                                getFilename());
                                    }
                              }

                              _cnt118++;
                        } while (true);
                  }
                  print_stmt_AST = (AST) currentAST.root;
                  print_stmt_AST =
                        (AST) astFactory.make(
                              (new ASTArray(2))
                                    .add(
                                          astFactory.create(
                                                PRINT_STMT,
                                                "PRINT_STMT"))
                                    .add(print_stmt_AST));
                  currentAST.root = print_stmt_AST;
                  currentAST.child =
                        print_stmt_AST != null
                              &&   print_stmt_AST.getFirstChild()    !=
null
```

```
                                    ? print_stmt_AST.getFirstChild()
                                    : print_stmt_AST;
                currentAST.advanceChildToEnd();
                print_stmt_AST = (AST) currentAST.root;
        } catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_2);
        }
        returnAST = print_stmt_AST;
    }

    public final void for_expr()
        throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST for_expr_AST = null;

        try { // for error handling
                numid_expr();
                astFactory.addASTChild(currentAST, returnAST);
                match(COMMA);
                numid_expr();
                astFactory.addASTChild(currentAST, returnAST);
                match(COMMA);
                AST tmp47_AST = null;
                tmp47_AST = astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp47_AST);
                match(ID);
                for_expr_AST = (AST) currentAST.root;
                for_expr_AST =
                      (AST) astFactory.make(
                            (new ASTArray(2))
                                  .add(
                                        astFactory.create(
```

```
                                                FOR_EXPR,
                                                "FOR_EXPR"))
                                    .add(for_expr_AST));
                  currentAST.root = for_expr_AST;
                  currentAST.child =
                        for_expr_AST != null
                              && for_expr_AST.getFirstChild() != null
                                    ? for_expr_AST.getFirstChild()
                                    : for_expr_AST;
                  currentAST.advanceChildToEnd();
                  for_expr_AST = (AST) currentAST.root;
            } catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_4);
            }
            returnAST = for_expr_AST;
      }

      public final void forloop_stmt()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST forloop_stmt_AST = null;

            try { // for error handling
                  match(LBRACE);
                  {
                        int _cnt130 = 0;
                        _loop130 : do {
                              if ((LA(1) == NL)) {
                                    match(NL);
                              } else {
                                    if (_cnt130 >= 1) {
                                          break _loop130;
```

```
                                    } else {
                                          throw                    new
NoViableAltException(
                                                LT(1),
                                                getFilename());
                                    }
                              }

                              _cnt130++;
                        } while (true);
                  }
                  {
                        _loop132 : do {
                              if ((_tokenSet_5.member(LA(1)))) {
                                    statement();
                                    astFactory.addASTChild(
                                          currentAST,
                                          returnAST);
                              } else {
                                    break _loop132;
                              }

                        } while (true);
                  }
                  match(RBRACE);
                  {
                        int _cnt134 = 0;
                        _loop134 : do {
                              if ((LA(1) == NL)) {
                                    match(NL);
                              } else {
                                    if (_cnt134 >= 1) {
                                          break _loop134;
                                    } else {
                                          throw                    new
NoViableAltException(
```

```
                                          LT(1),
                                          getFilename());
                                    }
                              }

                              _cnt134++;
                        } while (true);
                  }
                  forloop_stmt_AST = (AST) currentAST.root;
                  forloop_stmt_AST =
                        (AST) astFactory.make(
                              (new ASTArray(2))
                                    .add(
                                          astFactory.create(
                                                FORLOOP_STMT,
                                                "FORLOOP_STMT"))
                                    .add(forloop_stmt_AST));
                  currentAST.root = forloop_stmt_AST;
                  currentAST.child =
                        forloop_stmt_AST != null
                              && forloop_stmt_AST.getFirstChild()
                                    != null
                                    ? forloop_stmt_AST.getFirstChild()
                                    : forloop_stmt_AST;
                  currentAST.advanceChildToEnd();
                  forloop_stmt_AST = (AST) currentAST.root;
            } catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_2);
            }
            returnAST = forloop_stmt_AST;
      }

      public final void logic_expr()
            throws RecognitionException, TokenStreamException {
```

```
        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST logic_expr_AST = null;

        try { // for error handling
              logic_term();
              astFactory.addASTChild(currentAST, returnAST);
              {
                    _loop154 : do {
                          if ((LA(1) == OR)) {
                                AST tmp52_AST = null;
                                tmp52_AST =
                                      astFactory.create(LT(1));
                                astFactory.makeASTRoot(
                                      currentAST,
                                      tmp52_AST);
                                match(OR);
                                logic_term();
                                astFactory.addASTChild(
                                      currentAST,
                                      returnAST);
                          } else {
                                break _loop154;
                          }

                    } while (true);
              }
              logic_expr_AST = (AST) currentAST.root;
        } catch (RecognitionException ex) {
              reportError(ex);
              consume();
              consumeUntil(_tokenSet_4);
        }
        returnAST = logic_expr_AST;
  }
```

```
      public final void block_stmt()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST block_stmt_AST = null;

            try { // for error handling
                  match(LBRACE);
                  {
                        int _cnt87 = 0;
                        _loop87 : do {
                              if ((LA(1) == NL)) {
                                    match(NL);
                              } else {
                                    if (_cnt87 >= 1) {
                                          break _loop87;
                                    } else {
                                          throw                    new
NoViableAltException(
                                                LT(1),
                                                getFilename());
                                    }
                              }

                              _cnt87++;
                        } while (true);
                  }
                  {
                        _loop89 : do {
                              switch (LA(1)) {
                                    case ID :
                                    case TYPE :
                                    case INSTRUMENT :
                                    case LITERAL_for :
```

```
                                    case LITERAL_foreach :
                                    case LITERAL_if :
                                    case LITERAL_add :
                                    case LITERAL_print :
                                          {
                                                statement();

      astFactory.addASTChild(
                                                      currentAST,
                                                      returnAST);
                                                break;
                                          }
                                    case LITERAL_break :
                                          {
                                                break_stmt();

      astFactory.addASTChild(
                                                      currentAST,
                                                      returnAST);
                                                break;
                                          }
                                    default :
                                          {
                                                break _loop89;
                                          }
                              }
                        } while (true);
                  }
                  match(RBRACE);
                  {
                        int _cnt91 = 0;
                        _loop91 : do {
                              if ((LA(1) == NL)) {
                                    match(NL);
                              } else {
                                    if (_cnt91 >= 1) {
```

```
                                        break _loop91;
                                  } else {
                                        throw                    new
NoViableAltException(
                                              LT(1),
                                              getFilename());
                                  }
                            }

                            _cnt91++;
                      } while (true);
                }
                block_stmt_AST = (AST) currentAST.root;
                block_stmt_AST =
                      (AST) astFactory.make(
                            (new ASTArray(2))
                                  .add(
                                        astFactory.create(
                                              BLOCK_STMT,
                                              "BLOCK_STMT"))
                                  .add(block_stmt_AST));
                currentAST.root = block_stmt_AST;
                currentAST.child =
                      block_stmt_AST != null
                            &&   block_stmt_AST.getFirstChild()   !=
null
                                  ? block_stmt_AST.getFirstChild()
                                  : block_stmt_AST;
                currentAST.advanceChildToEnd();
                block_stmt_AST = (AST) currentAST.root;
          } catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_6);
          }
          returnAST = block_stmt_AST;
```

```
	}

	public final void break_stmt()
		throws RecognitionException, TokenStreamException {

		returnAST = null;
		ASTPair currentAST = new ASTPair();
		AST break_stmt_AST = null;

		try { // for error handling
			AST tmp57_AST = null;
			tmp57_AST = astFactory.create(LT(1));
			astFactory.makeASTRoot(currentAST, tmp57_AST);
			match(LITERAL_break);
			{
				int _cnt94 = 0;
				_loop94 : do {
					if ((LA(1) == NL)) {
						match(NL);
					} else {
						if (_cnt94 >= 1) {
							break _loop94;
						} else {
							throw                    new
NoViableAltException(
                                                LT(1),
                                                getFilename());
						}
					}

					_cnt94++;
				} while (true);
			}
			break_stmt_AST = (AST) currentAST.root;
		} catch (RecognitionException ex) {
			reportError(ex);
```

```
                consume();
                consumeUntil(_tokenSet_7);
          }
          returnAST = break_stmt_AST;
     }

     public final void return_stmt()
          throws RecognitionException, TokenStreamException {

          returnAST = null;
          ASTPair currentAST = new ASTPair();
          AST return_stmt_AST = null;

          try { // for error handling
                AST tmp59_AST = null;
                tmp59_AST = astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp59_AST);
                match(LITERAL_return);
                return_expr();
                astFactory.addASTChild(currentAST, returnAST);
                {
                      int _cnt97 = 0;
                      _loop97 : do {
                            if ((LA(1) == NL)) {
                                  match(NL);
                            } else {
                                  if (_cnt97 >= 1) {
                                        break _loop97;
                                  } else {
                                        throw                    new
NoViableAltException(
                                              LT(1),
                                              getFilename());
                                  }
                            }
```

```
                        _cnt97++;
                  } while (true);
            }
            return_stmt_AST = (AST) currentAST.root;
      } catch (RecognitionException ex) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_8);
      }
      returnAST = return_stmt_AST;
}

public final void return_expr()
      throws RecognitionException, TokenStreamException {

      returnAST = null;
      ASTPair currentAST = new ASTPair();
      AST return_expr_AST = null;

      try { // for error handling
            if ((LA(1) == ID) && (LA(2) == NL)) {
                  AST tmp61_AST = null;
                  tmp61_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(
                        currentAST,
                        tmp61_AST);
                  match(ID);
                  return_expr_AST = (AST) currentAST.root;
            } else if (
                  (LA(1) == ID) && (LA(2) == LPAREN)) {
                  funct_call_expr();
                  astFactory.addASTChild(
                        currentAST,
                        returnAST);
                  return_expr_AST = (AST) currentAST.root;
            } else {
```

```
                        throw new NoViableAltException(
                              LT(1),
                              getFilename());
                  }

            } catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_9);
            }
            returnAST = return_expr_AST;
      }

      public final void declare_stmt()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST declare_stmt_AST = null;

            try { // for error handling
                  AST tmp62_AST = null;
                  tmp62_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp62_AST);
                  match(TYPE);
                  AST tmp63_AST = null;
                  tmp63_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp63_AST);
                  match(ID);
                  declare_stmt_AST = (AST) currentAST.root;
                  declare_stmt_AST =
                        (AST) astFactory.make(
                              (new ASTArray(2))
                                    .add(
                                          astFactory.create(
                                                DEC_EXPR,
```

```
                                              "DEC_EXPR"))
                                    .add(declare_stmt_AST));
                  currentAST.root = declare_stmt_AST;
                  currentAST.child =
                        declare_stmt_AST != null
                              && declare_stmt_AST.getFirstChild()
                                    != null
                                    ? declare_stmt_AST.getFirstChild()
                                    : declare_stmt_AST;
                  currentAST.advanceChildToEnd();
                  declare_stmt_AST = (AST) currentAST.root;
            } catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_10);
            }
            returnAST = declare_stmt_AST;
      }

      public final void right_expr()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST right_expr_AST = null;

            try { // for error handling
                  switch (LA(1)) {
                        case ID :
                        case LPAREN :
                        case PLUS :
                        case MINUS :
                        case NUM :
                              {
                                    multdiv_expr();
                                    astFactory.addASTChild(
```

```
                                        currentAST,
                                        returnAST);
                                  {
                                        _loop182 : do {
                                              if ((LA(1) == PLUS
                                                    ||    LA(1)    ==
MINUS)) {
                                                    {
                                                          switch
(LA(1)) {
                                                                case
PLUS :
      {
            AST tmp64_AST =
                  null;
            tmp64_AST =
                  astFactory
                        .create(
                        LT(1));
            astFactory
                  .makeASTRoot(
                  currentAST,
                  tmp64_AST);
            match(PLUS);
```

```
            break;

      }
                                                                  case
MINUS :

      {

            AST tmp65_AST =

                  null;

            tmp65_AST =

                  astFactory

                        .create(

                        LT(1));

            astFactory

                  .makeASTRoot(

                  currentAST,

                  tmp65_AST);

            match(MINUS);

            break;

      }

      default :
```

```
{

      throw new NoViableAltException(

            LT(1),

            getFilename());

}
                                                      }
                                                }
                                                multdiv_expr();

astFactory.addASTChild(

currentAST,

returnAST);
                                          } else {
                                                break _loop182;
                                          }

                                    } while (true);
                              }
                              right_expr_AST =
                                    (AST) currentAST.root;
                              break;
                        }
                  case NOTE :
                        {
                              {
                                    int _cnt184 = 0;
                                    _loop184 : do {
                                          if ((LA(1) == NOTE)) {
                                                AST  tmp66_AST  =
```

```
null;
                                                        tmp66_AST =

      astFactory.create(

      LT(1));

      astFactory.addASTChild(

      currentAST,

      tmp66_AST);
                                                        match(NOTE);
                                                  } else {
                                                        if   (_cnt184   >=
1) {
                                                              break
_loop184;
                                                        } else {
                                                              throw   new
NoViableAltException(

      LT(1),

      getFilename());
                                                        }
                                                  }

                                                        _cnt184++;
                                            } while (true);
                                      }
                                      right_expr_AST =
                                            (AST) currentAST.root;
                                      right_expr_AST =
                                            (AST) astFactory.make(
                                                  (new ASTArray(2))
```

```
                                                      .add(

      astFactory.create(

      RIGHT_EXPR,

      "RIGHT_EXPR"))

      .add(right_expr_AST));
                                    currentAST.root = right_expr_AST;
                                    currentAST.child =
                                          right_expr_AST != null
                                                && right_expr_AST
                                                      .getFirstChild()
                                                      != null
                                                      ? right_expr_AST

      .getFirstChild()
                                                      :
right_expr_AST;
                                    currentAST.advanceChildToEnd();
                                    right_expr_AST =
                                          (AST) currentAST.root;
                                    break;
                              }
                        default :
                              {
                                    throw new NoViableAltException(
                                          LT(1),
                                          getFilename());
                              }
                  }
            } catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_9);
```

```
        }
        returnAST = right_expr_AST;
    }

    public final void left_expr()
        throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST left_expr_AST = null;

        try { // for error handling
            id_expr();
            astFactory.addASTChild(currentAST, returnAST);
            left_expr_AST = (AST) currentAST.root;
            left_expr_AST =
                (AST) astFactory.make(
                    (new ASTArray(2))
                        .add(
                            astFactory.create(
                                LEFT_EXPR,
                                "LEFT_EXPR"))
                        .add(left_expr_AST));
            currentAST.root = left_expr_AST;
            currentAST.child =
                left_expr_AST != null
                    && left_expr_AST.getFirstChild() != null
                        ? left_expr_AST.getFirstChild()
                        : left_expr_AST;
            currentAST.advanceChildToEnd();
            left_expr_AST = (AST) currentAST.root;
        } catch (RecognitionException ex) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_11);
        }
```

```
            returnAST = left_expr_AST;
      }

      public final void funct_call_expr()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST funct_call_expr_AST = null;

            try { // for error handling
                  AST tmp67_AST = null;
                  tmp67_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp67_AST);
                  match(ID);
                  match(LPAREN);
                  param_expr();
                  astFactory.addASTChild(currentAST, returnAST);
                  match(RPAREN);
                  funct_call_expr_AST = (AST) currentAST.root;
                  funct_call_expr_AST =
                        (AST) astFactory.make(
                              (new ASTArray(2))
                                    .add(
                                          astFactory.create(
                                                FUNCT_CALL_EXPR,
                                                "FUNCT_CALL_EXPR"))
                                    .add(funct_call_expr_AST));
                  currentAST.root = funct_call_expr_AST;
                  currentAST.child =
                        funct_call_expr_AST != null
                              && funct_call_expr_AST.getFirstChild()
                                    != null
                                    ?
funct_call_expr_AST.getFirstChild()
                                    : funct_call_expr_AST;
```

```
                currentAST.advanceChildToEnd();
                funct_call_expr_AST = (AST) currentAST.root;
          } catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_12);
          }
          returnAST = funct_call_expr_AST;
    }

    public final void param_expr()
          throws RecognitionException, TokenStreamException {

          returnAST = null;
          ASTPair currentAST = new ASTPair();
          AST param_expr_AST = null;

          try { // for error handling
                {
                      switch (LA(1)) {
                            case ID :
                            case MINUS :
                            case NUM :
                                  {
                                        numid_expr();
                                        astFactory.addASTChild(
                                              currentAST,
                                              returnAST);
                                        {
                                              _loop139 : do {
                                                    if   ((LA(1)   ==
COMMA)) {

    match(COMMA);

    numid_expr();
```

```
                                                                  astFactory

      .addASTChild(

      currentAST,

      returnAST);
                                                        } else {
                                                              break
_loop139;
                                                        }

                                                  } while (true);
                                             }
                                             break;
                                       }
                                 case RPAREN :
                                       {
                                             break;
                                       }
                                 default :
                                       {
                                             throw                    new
NoViableAltException(
                                                   LT(1),
                                                   getFilename());
                                       }
                           }
                     }
                     param_expr_AST = (AST) currentAST.root;
                     param_expr_AST =
                           (AST) astFactory.make(
                                 (new ASTArray(2))
                                       .add(
                                             astFactory.create(
                                                   PARAM_EXPR,
```

```
                                                "PARAM_EXPR"))
                                    .add(param_expr_AST));
                  currentAST.root = param_expr_AST;
                  currentAST.child =
                        param_expr_AST != null
                              &&    param_expr_AST.getFirstChild()    !=
null
                                    ? param_expr_AST.getFirstChild()
                                    : param_expr_AST;
                  currentAST.advanceChildToEnd();
                  param_expr_AST = (AST) currentAST.root;
            } catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_4);
            }
            returnAST = param_expr_AST;
      }

      public final void dec_param_expr()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST dec_param_expr_AST = null;

            try { // for error handling
                  {
                        switch (LA(1)) {
                              case TYPE :
                                    {
                                          declare_stmt();
                                          astFactory.addASTChild(
                                                currentAST,
                                                returnAST);
                                          {
```

```
                                                _loop143 : do {
                                                      if   ((LA(1)   ==
COMMA)) {

      match(COMMA);

      declare_stmt();
                                                            astFactory

      .addASTChild(

      currentAST,

      returnAST);
                                                      } else {
                                                            break
_loop143;
                                                      }

                                                } while (true);
                                          }
                                          break;
                                    }
                              case RPAREN :
                                    {
                                          break;
                                    }
                              default :
                                    {
                                          throw                    new
NoViableAltException(
                                                LT(1),
                                                getFilename());
                                    }
                        }
                  }
```

```
                  dec_param_expr_AST = (AST) currentAST.root;
                  dec_param_expr_AST =
                        (AST) astFactory.make(
                              (new ASTArray(2))
                                    .add(
                                          astFactory.create(
                                                DEC_PARAM_EXPR,
                                                "DEC_PARAM_EXPR"))
                                    .add(dec_param_expr_AST));
                  currentAST.root = dec_param_expr_AST;
                  currentAST.child =
                        dec_param_expr_AST != null
                              && dec_param_expr_AST.getFirstChild()
                                    != null
                                    ?
dec_param_expr_AST.getFirstChild()
                                    : dec_param_expr_AST;
                  currentAST.advanceChildToEnd();
                  dec_param_expr_AST = (AST) currentAST.root;
            } catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_4);
            }
            returnAST = dec_param_expr_AST;
      }

      public final void funct_stmt()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST funct_stmt_AST = null;

            try { // for error handling
                  match(LBRACE);
```

```
                  {
                        int _cnt146 = 0;
                        _loop146 : do {
                              if ((LA(1) == NL)) {
                                    match(NL);
                              } else {
                                    if (_cnt146 >= 1) {
                                          break _loop146;
                                    } else {
                                          throw                    new
NoViableAltException(
                                                LT(1),
                                                getFilename());
                                    }
                              }

                              _cnt146++;
                        } while (true);
                  }
                  {
                        _loop148 : do {
                              if ((_tokenSet_5.member(LA(1)))) {
                                    statement();
                                    astFactory.addASTChild(
                                          currentAST,
                                          returnAST);
                              } else {
                                    break _loop148;
                              }

                        } while (true);
                  }
                  {
                        return_stmt();
                        astFactory.addASTChild(
                              currentAST,
```

```
                              returnAST);
                  }
                  match(RBRACE);
                  {
                        int _cnt151 = 0;
                        _loop151 : do {
                              if ((LA(1) == NL)) {
                                    match(NL);
                              } else {
                                    if (_cnt151 >= 1) {
                                          break _loop151;
                                    } else {
                                          throw                    new
NoViableAltException(
                                                LT(1),
                                                getFilename());
                                    }
                              }

                              _cnt151++;
                        } while (true);
                  }
                  funct_stmt_AST = (AST) currentAST.root;
                  funct_stmt_AST =
                        (AST) astFactory.make(
                              (new ASTArray(2))
                                    .add(
                                          astFactory.create(
                                                FUNCT_STMT,
                                                "FUNCT_STMT"))
                                    .add(funct_stmt_AST));
                  currentAST.root = funct_stmt_AST;
                  currentAST.child =
                        funct_stmt_AST != null
                              &&   funct_stmt_AST.getFirstChild()    !=
null
```

```
                                    ? funct_stmt_AST.getFirstChild()
                                    : funct_stmt_AST;
                  currentAST.advanceChildToEnd();
                  funct_stmt_AST = (AST) currentAST.root;
            } catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_3);
            }
            returnAST = funct_stmt_AST;
      }

      public final void numid_expr()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST numid_expr_AST = null;

            try { // for error handling
                  switch (LA(1)) {
                        case MINUS :
                              {
                                    match(MINUS);
                                    AST tmp77_AST = null;
                                    tmp77_AST =
                                          astFactory.create(LT(1));
                                    astFactory.addASTChild(
                                          currentAST,
                                          tmp77_AST);
                                    match(NUM);
                                    numid_expr_AST =
                                          (AST) currentAST.root;
                                    numid_expr_AST =
                                          (AST) astFactory.make(
                                                (new ASTArray(2))
```

```
                                                .add(

      astFactory.create(

      UMINUS,

      "UMINUS"))

      .add(numid_expr_AST));
                                    currentAST.root = numid_expr_AST;
                                    currentAST.child =
                                          numid_expr_AST != null
                                                && numid_expr_AST
                                                      .getFirstChild()
                                                      != null
                                                      ? numid_expr_AST

      .getFirstChild()
                                                      :
numid_expr_AST;
                                    currentAST.advanceChildToEnd();
                                    numid_expr_AST =
                                          (AST) currentAST.root;
                                    break;
                              }
                        case NUM :
                              {
                                    AST tmp78_AST = null;
                                    tmp78_AST =
                                          astFactory.create(LT(1));
                                    astFactory.addASTChild(
                                          currentAST,
                                          tmp78_AST);
                                    match(NUM);
                                    numid_expr_AST =
                                          (AST) currentAST.root;
```

```
                                    break;
                              }
                        case ID :
                              {
                                    AST tmp79_AST = null;
                                    tmp79_AST =
                                          astFactory.create(LT(1));
                                    astFactory.addASTChild(
                                          currentAST,
                                          tmp79_AST);
                                    match(ID);
                                    numid_expr_AST =
                                          (AST) currentAST.root;
                                    break;
                              }
                        default :
                              {
                                    throw new NoViableAltException(
                                          LT(1),
                                          getFilename());
                              }
                  }
            } catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_13);
            }
            returnAST = numid_expr_AST;
      }

      public final void logic_term()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST logic_term_AST = null;
```

```
        try { // for error handling
              logic_point();
              astFactory.addASTChild(currentAST, returnAST);
              {
                    _loop157 : do {
                          if ((LA(1) == AND)) {
                                AST tmp80_AST = null;
                                tmp80_AST =
                                      astFactory.create(LT(1));
                                astFactory.makeASTRoot(
                                      currentAST,
                                      tmp80_AST);
                                match(AND);
                                logic_point();
                                astFactory.addASTChild(
                                      currentAST,
                                      returnAST);
                          } else {
                                break _loop157;
                          }

                    } while (true);
              }
              logic_term_AST = (AST) currentAST.root;
        } catch (RecognitionException ex) {
              reportError(ex);
              consume();
              consumeUntil(_tokenSet_14);
        }
        returnAST = logic_term_AST;
  }

  public final void logic_point()
        throws RecognitionException, TokenStreamException {
```

```
            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST logic_point_AST = null;

            try { // for error handling
                  {
                        switch (LA(1)) {
                              case NOT :
                                    {
                                          AST tmp81_AST = null;
                                          tmp81_AST =

      astFactory.create(LT(1));
                                          astFactory.makeASTRoot(
                                                currentAST,
                                                tmp81_AST);
                                          match(NOT);
                                          break;
                                    }
                              case ID :
                              case LPAREN :
                              case PLUS :
                              case MINUS :
                              case NUM :
                                    {
                                          break;
                                    }
                              default :
                                    {
                                          throw                    new
NoViableAltException(
                                                LT(1),
                                                getFilename());
                                    }
                        }
                  }
```

```
                relational_expr();
                astFactory.addASTChild(currentAST, returnAST);
                logic_point_AST = (AST) currentAST.root;
          } catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_15);
          }
          returnAST = logic_point_AST;
     }

     public final void relational_expr()
          throws RecognitionException, TokenStreamException {

          returnAST = null;
          ASTPair currentAST = new ASTPair();
          AST relational_expr_AST = null;

          try { // for error handling
                addsub_expr();
                astFactory.addASTChild(currentAST, returnAST);
                {
                      switch (LA(1)) {
                            case GE :
                            case LE :
                            case GT :
                            case LT :
                            case EQ :
                            case NEQ :
                                  {
                                        {
                                              switch (LA(1)) {
                                                    case GE :
                                                          {
                                                                AST
tmp82_AST =
```

```
      null;
      tmp82_AST =
      astFactory
            .create(
            LT(1));
      astFactory
      .makeASTRoot(
      currentAST,
      tmp82_AST);
      match(GE);
      break;
                                                      }
                                                case LE :
                                                      {
                                                            AST
tmp83_AST =
      null;
      tmp83_AST =
      astFactory
            .create(
```

```
            LT(1));

      astFactory

      .makeASTRoot(

      currentAST,

      tmp83_AST);

      match(LE);

      break;
                                                      }
                                                case GT :
                                                      {
                                                            AST
tmp84_AST =

      null;

      tmp84_AST =

      astFactory

            .create(

            LT(1));

      astFactory

      .makeASTRoot(

      currentAST,

      tmp84_AST);
```

```
      match(GT);
      break;
                                                        }
                                                  case LT :
                                                        {
                                                              AST
tmp85_AST =
      null;
      tmp85_AST =
      astFactory
            .create(
            LT(1));
      astFactory
      .makeASTRoot(
      currentAST,
      tmp85_AST);
      match(LT);
      break;
                                                        }
                                                  case EQ :
                                                        {
                                                              AST
tmp86_AST =
```

```
      null;
      tmp86_AST =
      astFactory
            .create(
            LT(1));
      astFactory
      .makeASTRoot(
      currentAST,
      tmp86_AST);
      match(EQ);
      break;
                                                      }
                                                case NEQ :
                                                      {
                                                            AST
tmp87_AST =
      null;
      tmp87_AST =
      astFactory
            .create(
```

```
            LT(1));

      astFactory

      .makeASTRoot(

      currentAST,

      tmp87_AST);

      match(NEQ);

      break;
                                                            }
                                                      default :
                                                            {

      throw new NoViableAltException(

      LT(1),

      getFilename());
                                                            }
                                                      }
                                                }
                                                addsub_expr();
                                                astFactory.addASTChild(
                                                      currentAST,
                                                      returnAST);
                                                break;
                                          }
                                    case RPAREN :
                                    case AND :
                                    case OR :
                                          {
                                                break;
```

```
                                    }
                              default :
                                    {
                                          throw                    new
NoViableAltException(
                                                LT(1),
                                                getFilename());
                                    }
                        }
                  }
                  relational_expr_AST = (AST) currentAST.root;
            } catch (RecognitionException ex) {
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_15);
            }
            returnAST = relational_expr_AST;
      }

      public final void addsub_expr()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST addsub_expr_AST = null;

            try { // for error handling
                  multdiv_expr();
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                        _loop166 : do {
                              if ((LA(1) == PLUS
                                    || LA(1) == MINUS)) {
                                    {
                                          switch (LA(1)) {
                                                case PLUS :
```

```
                                                        {
                                                              AST
tmp88_AST =
      null;
                                                              tmp88_AST
=
      astFactory
      .create(
      LT(1));
                                                              astFactory
      .makeASTRoot(
      currentAST,
      tmp88_AST);
      match(PLUS);
                                                              break;
                                                        }
                                                  case MINUS :
                                                        {
                                                              AST
tmp89_AST =
      null;
                                                              tmp89_AST
=
      astFactory
      .create(
```

```
      LT(1));
                                                                  astFactory
      .makeASTRoot(
      currentAST,
      tmp89_AST);
      match(MINUS);
                                                                  break;
                                                            }
                                                      default :
                                                            {
                                                                  throw   new
NoViableAltException(
      LT(1),
      getFilename());
                                                            }
                                          }
                                    }
                                    multdiv_expr();
                                    astFactory.addASTChild(
                                          currentAST,
                                          returnAST);
                              } else {
                                    break _loop166;
                              }

                        } while (true);
                  }
                  addsub_expr_AST = (AST) currentAST.root;
            } catch (RecognitionException ex) {
```

```
                  reportError(ex);
                  consume();
                  consumeUntil(_tokenSet_16);
            }
            returnAST = addsub_expr_AST;
      }

      public final void multdiv_expr()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST multdiv_expr_AST = null;

            try { // for error handling
                  math_term();
                  astFactory.addASTChild(currentAST, returnAST);
                  {
                        _loop170 : do {
                              if ((LA(1) == MULT
                                    || LA(1) == DIV
                                    || LA(1) == MOD)) {
                                    {
                                          switch (LA(1)) {
                                                case MULT :
                                                      {
                                                            AST
tmp90_AST =

      null;
                                                            tmp90_AST
=

      astFactory

      .create(
```

```
      LT(1));
                                                              astFactory
      .makeASTRoot(
      currentAST,
      tmp90_AST);
      match(MULT);
                                                              break;
                                                        }
                                                  case DIV :
                                                        {
                                                              AST
tmp91_AST =
      null;
                                                              tmp91_AST
=
      astFactory
      .create(
      LT(1));
                                                              astFactory
      .makeASTRoot(
      currentAST,
      tmp91_AST);
      match(DIV);
```

```
                                                                    break;
                                                                }
                                                            case MOD :
                                                                {
                                                                    AST
tmp92_AST =
      null;
                                                                    tmp92_AST
=
      astFactory
      .create(
      LT(1));
                                                                    astFactory
      .makeASTRoot(
      currentAST,
      tmp92_AST);
      match(MOD);
                                                                    break;
                                                                }
                                                            default :
                                                                {
                                                                    throw   new
NoViableAltException(
      LT(1),
      getFilename());
                                                                }
```

```
                                    }
                                }
                                math_term();
                                astFactory.addASTChild(
                                    currentAST,
                                    returnAST);
                            } else {
                                break _loop170;
                            }

                        } while (true);
                    }
                    multdiv_expr_AST = (AST) currentAST.root;
                } catch (RecognitionException ex) {
                    reportError(ex);
                    consume();
                    consumeUntil(_tokenSet_17);
                }
                returnAST = multdiv_expr_AST;
            }

            public final void math_term()
                throws RecognitionException, TokenStreamException {

                returnAST = null;
                ASTPair currentAST = new ASTPair();
                AST math_term_AST = null;

                try { // for error handling
                    switch (LA(1)) {
                        case PLUS :
                            {
                                match(PLUS);
                                math_point();
                                astFactory.addASTChild(
                                    currentAST,
```

```
                                    returnAST);
                              math_term_AST =
                                    (AST) currentAST.root;
                              math_term_AST =
                                    (AST) astFactory.make(
                                          (new ASTArray(2))
                                                .add(

      astFactory.create(

      PLUS_MATH_TERM,

      "PLUS_MATH_TERM"))

      .add(math_term_AST));
                              currentAST.root = math_term_AST;
                              currentAST.child =
                                    math_term_AST != null
                                          && math_term_AST
                                                .getFirstChild()
                                                != null
                                                ? math_term_AST

      .getFirstChild()
                                                : math_term_AST;
                              currentAST.advanceChildToEnd();
                              math_term_AST =
                                    (AST) currentAST.root;
                              break;
                        }
                  case MINUS :
                        {
                              match(MINUS);
                              math_point();
                              astFactory.addASTChild(
                                    currentAST,
```

```
                                    returnAST);
                              math_term_AST =
                                    (AST) currentAST.root;
                              math_term_AST =
                                    (AST) astFactory.make(
                                          (new ASTArray(2))
                                                .add(

      astFactory.create(

      MINUS_MATH_TERM,

      "MINUS_MATH_TERM"))

      .add(math_term_AST));
                              currentAST.root = math_term_AST;
                              currentAST.child =
                                    math_term_AST != null
                                          && math_term_AST
                                                .getFirstChild()
                                                != null
                                                ? math_term_AST

      .getFirstChild()
                                                : math_term_AST;
                              currentAST.advanceChildToEnd();
                              math_term_AST =
                                    (AST) currentAST.root;
                              break;
                        }
                  case ID :
                  case LPAREN :
                  case NUM :
                        {
                              math_point();
                              astFactory.addASTChild(
```

```
                                    currentAST,
                                    returnAST);
                              math_term_AST =
                                    (AST) currentAST.root;
                              math_term_AST =
                                    (AST) astFactory.make(
                                          (new ASTArray(2))
                                                .add(

      astFactory.create(

      MATH_TERM,

      "MATH_TERM"))

      .add(math_term_AST));
                              currentAST.root = math_term_AST;
                              currentAST.child =
                                    math_term_AST != null
                                          && math_term_AST
                                                .getFirstChild()
                                                != null
                                                ? math_term_AST

      .getFirstChild()
                                                : math_term_AST;
                              currentAST.advanceChildToEnd();
                              math_term_AST =
                                    (AST) currentAST.root;
                              break;
                        }
                  default :
                        {
                              throw new NoViableAltException(
                                    LT(1),
                                    getFilename());
```

```
                        }
                }
        } catch (RecognitionException ex) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_12);
        }
        returnAST = math_term_AST;
    }

    public final void math_point()
        throws RecognitionException, TokenStreamException {

        returnAST = null;
        ASTPair currentAST = new ASTPair();
        AST math_point_AST = null;

        try { // for error handling
                switch (LA(1)) {
                        case NUM :
                                {
                                        AST tmp95_AST = null;
                                        tmp95_AST =
                                                astFactory.create(LT(1));
                                        astFactory.addASTChild(
                                                currentAST,
                                                tmp95_AST);
                                        match(NUM);
                                        math_point_AST =
                                                (AST) currentAST.root;
                                        break;
                                }
                        case LPAREN :
                                {
                                        match(LPAREN);
                                        logic_expr();
```

```
                                    astFactory.addASTChild(
                                          currentAST,
                                          returnAST);
                                    match(RPAREN);
                                    math_point_AST =
                                          (AST) currentAST.root;
                                    break;
                              }
                        default :
                              if ((LA(1) == ID)
                                    && (_tokenSet_18.member(LA(2)))) {
                                    id_expr();
                                    astFactory.addASTChild(
                                          currentAST,
                                          returnAST);
                                    math_point_AST =
                                          (AST) currentAST.root;
                              } else if (
                                    (LA(1) == ID)
                                          && (LA(2) == LPAREN)) {
                                    funct_call_expr();
                                    astFactory.addASTChild(
                                          currentAST,
                                          returnAST);
                                    math_point_AST =
                                          (AST) currentAST.root;
                              } else {
                                    throw new NoViableAltException(
                                          LT(1),
                                          getFilename());
                              }
                        }
                  } catch (RecognitionException ex) {
                        reportError(ex);
                        consume();
                        consumeUntil(_tokenSet_12);
```

```
            }
            returnAST = math_point_AST;
      }

      public final void id_expr()
            throws RecognitionException, TokenStreamException {

            returnAST = null;
            ASTPair currentAST = new ASTPair();
            AST id_expr_AST = null;

            try { // for error handling
                  AST tmp98_AST = null;
                  tmp98_AST = astFactory.create(LT(1));
                  astFactory.addASTChild(currentAST, tmp98_AST);
                  match(ID);
                  {
                        switch (LA(1)) {
                              case DOT :
                                    {
                                          AST tmp99_AST = null;
                                          tmp99_AST =

      astFactory.create(LT(1));
                                          astFactory.addASTChild(
                                                currentAST,
                                                tmp99_AST);
                                          match(DOT);
                                          {
                                                switch (LA(1)) {
                                                      case
LITERAL_pitch :
                                                                  {
                                                                        AST
tmp100_AST =
```

```
      null;

      tmp100_AST =

      astFactory

            .create(

            LT(1));

      astFactory

      .addASTChild(

      currentAST,

      tmp100_AST);

      match(LITERAL_pitch);

      break;
                                                                  }
                                                            case
LITERAL_length :
                                                                  {
                                                                        AST
tmp101_AST =

      null;

      tmp101_AST =

      astFactory

            .create(
```

```
            LT(1));

      astFactory

      .addASTChild(

      currentAST,

      tmp101_AST);

      match(LITERAL_length);

      break;
                                                            }
                                                      case
LITERAL_volume :
                                                            {
                                                                  AST
tmp102_AST =

      null;

      tmp102_AST =

      astFactory

            .create(

            LT(1));

      astFactory

      .addASTChild(

      currentAST,
```

```
      tmp102_AST);

      match(LITERAL_volume);

      break;
                                                            }
                                                      default :
                                                            {

      throw new NoViableAltException(

      LT(1),

      getFilename());
                                                            }
                                                }
                                          }
                                          break;
                                    }
                              case NL :
                              case RPAREN :
                              case MULT :
                              case PLUS :
                              case MINUS :
                              case DIV :
                              case MOD :
                              case ASGN :
                              case GE :
                              case LE :
                              case GT :
                              case LT :
                              case EQ :
                              case NEQ :
                              case AND :
                              case OR :
                                    {
```

```
                                    break;
                              }
                        default :
                              {
                                    throw                    new
NoViableAltException(
                                          LT(1),
                                          getFilename());
                              }
                  }
            }
            id_expr_AST = (AST) currentAST.root;
            id_expr_AST =
                  (AST) astFactory.make(
                        (new ASTArray(2))
                              .add(
                                    astFactory.create(
                                          ID_EXPR,
                                          "ID_EXPR"))
                              .add(id_expr_AST));
            currentAST.root = id_expr_AST;
            currentAST.child =
                  id_expr_AST != null
                        && id_expr_AST.getFirstChild() != null
                              ? id_expr_AST.getFirstChild()
                              : id_expr_AST;
            currentAST.advanceChildToEnd();
            id_expr_AST = (AST) currentAST.root;
      } catch (RecognitionException ex) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_19);
      }
      returnAST = id_expr_AST;
}
```

```
    public static final String[] _tokenNames =
        {
            "<0>",
            "EOF",
            "<2>",
            "NULL_TREE_LOOKAHEAD",
            "LOWER",
            "UPPER",
            "UNDER",
            "ALPHA",
            "DIGIT",
            "PITCH",
            "WS",
            "NL",
            "COMMENT",
            "ID",
            "TYPE",
            "LPAREN",
            "RPAREN",
            "LBRACE",
            "RBRACE",
            "COLON",
            "COMMA",
            "DOT",
            "MULT",
            "PLUS",
            "MINUS",
            "DIV",
            "MOD",
            "SHARP",
            "FLAT",
            "ASGN",
            "GE",
            "LE",
            "GT",
            "LT",
```

```
                    "EQ",
                    "NEQ",
                    "AND",
                    "OR",
                    "NOT",
                    "NUM",
                    "NOTE",
                    "STRING",
                    "INSTRUMENT",
                    "INSTRUMENT_NAME",
                    "STATEMENT",
                    "FORLOOP_STMT",
                    "PRINT_STMT",
                    "BLOCK_STMT",
                    "FUNCT_CALL_EXPR",
                    "FOR_EXPR",
                    "PARAM_EXPR",
                    "DEC_PARAM_EXPR",
                    "FUNCT_STMT",
                    "ID_EXPR",
                    "MATH_TERM",
                    "LEFT_EXPR",
                    "RIGHT_EXPR",
                    "NUMID_EXPR",
                    "UMINUS",
                    "DEC_EXPR",
                    "DEC_ASGN",
                    "LEFT_ASGN",
                    "PLUS_MATH_TERM",
                    "MINUS_MATH_TERM",
                    "\"for\"",
                    "\"foreach\"",
                    "\"if\"",
                    "\"else\"",
                    "\"break\"",
                    "\"return\"",
```

```
                "\"add\"",
                "\"print\"",
                "\"procedure\"",
                "\"function\"",
                "\"save\"",
                "\"playSong\"",
                "\"pitch\"",
                "\"length\"",
                "\"volume\"" };

    protected void buildTokenTypeASTClassMap() {
          tokenTypeToASTClassMap = null;
    };

    private static final long[] mk_tokenSet_0() {
          long[] data = { 2L, 0L };
          return data;
    }
    public static final BitSet _tokenSet_0 =
          new BitSet(mk_tokenSet_0());
    private static final long[] mk_tokenSet_1() {
          long[] data = { 4398046535680L, 967L, 0L, 0L };
          return data;
    }
    public static final BitSet _tokenSet_1 =
          new BitSet(mk_tokenSet_1());
    private static final long[] mk_tokenSet_2() {
          long[] data = { 4398046797826L, 3063L, 0L, 0L };
          return data;
    }
    public static final BitSet _tokenSet_2 =
          new BitSet(mk_tokenSet_2());
    private static final long[] mk_tokenSet_3() {
          long[] data = { 4398046535682L, 3015L, 0L, 0L };
          return data;
    }
```

```
    public static final BitSet _tokenSet_3 =
          new BitSet(mk_tokenSet_3());
    private static final long[] mk_tokenSet_4() {
          long[] data = { 65536L, 0L };
          return data;
    }
    public static final BitSet _tokenSet_4 =
          new BitSet(mk_tokenSet_4());
    private static final long[] mk_tokenSet_5() {
          long[] data = { 4398046535680L, 199L, 0L, 0L };
          return data;
    }
    public static final BitSet _tokenSet_5 =
          new BitSet(mk_tokenSet_5());
    private static final long[] mk_tokenSet_6() {
          long[] data = { 4398046797826L, 3071L, 0L, 0L };
          return data;
    }
    public static final BitSet _tokenSet_6 =
          new BitSet(mk_tokenSet_6());
    private static final long[] mk_tokenSet_7() {
          long[] data = { 4398046797824L, 215L, 0L, 0L };
          return data;
    }
    public static final BitSet _tokenSet_7 =
          new BitSet(mk_tokenSet_7());
    private static final long[] mk_tokenSet_8() {
          long[] data = { 262144L, 0L };
          return data;
    }
    public static final BitSet _tokenSet_8 =
          new BitSet(mk_tokenSet_8());
    private static final long[] mk_tokenSet_9() {
          long[] data = { 2048L, 0L };
          return data;
    }
```

```
    public static final BitSet _tokenSet_9 =
          new BitSet(mk_tokenSet_9());
    private static final long[] mk_tokenSet_10() {
          long[] data = { 537987072L, 0L };
          return data;
    }
    public static final BitSet _tokenSet_10 =
          new BitSet(mk_tokenSet_10());
    private static final long[] mk_tokenSet_11() {
          long[] data = { 536870912L, 0L };
          return data;
    }
    public static final BitSet _tokenSet_11 =
          new BitSet(mk_tokenSet_11());
    private static final long[] mk_tokenSet_12() {
          long[] data = { 273934256128L, 0L };
          return data;
    }
    public static final BitSet _tokenSet_12 =
          new BitSet(mk_tokenSet_12());
    private static final long[] mk_tokenSet_13() {
          long[] data = { 1114112L, 0L };
          return data;
    }
    public static final BitSet _tokenSet_13 =
          new BitSet(mk_tokenSet_13());
    private static final long[] mk_tokenSet_14() {
          long[] data = { 137439019008L, 0L };
          return data;
    }
    public static final BitSet _tokenSet_14 =
          new BitSet(mk_tokenSet_14());
    private static final long[] mk_tokenSet_15() {
          long[] data = { 206158495744L, 0L };
          return data;
    }
```

```
	public static final BitSet _tokenSet_15 =
		new BitSet(mk_tokenSet_15());
	private static final long[] mk_tokenSet_16() {
		long[] data = { 273804230656L, 0L };
		return data;
	}
	public static final BitSet _tokenSet_16 =
		new BitSet(mk_tokenSet_16());
	private static final long[] mk_tokenSet_17() {
		long[] data = { 273829398528L, 0L };
		return data;
	}
	public static final BitSet _tokenSet_17 =
		new BitSet(mk_tokenSet_17());
	private static final long[] mk_tokenSet_18() {
		long[] data = { 273936353280L, 0L };
		return data;
	}
	public static final BitSet _tokenSet_18 =
		new BitSet(mk_tokenSet_18());
	private static final long[] mk_tokenSet_19() {
		long[] data = { 274471127040L, 0L };
		return data;
	}
	public static final BitSet _tokenSet_19 =
		new BitSet(mk_tokenSet_19());

}
```

MTunesSequence.java

```
/*
 * Created on 2003/12/07
 */
import jm.music.data.Note;
import jm.music.data.Phrase;

/**
 * @author Hideki Sano, Huitao Sheng
 */
public class MTunesSequence extends MTunesVariable {

	private Phrase phrase;

	public MTunesSequence(String name, Phrase phrase) {
		super(MTunesVariable.SEQUENCE, name);
		this.phrase = phrase;
	}

	public final void setValue(Phrase phrase) {
		this.phrase = phrase;
	}

	public final void setValue(Note note) {
		this.phrase = new Phrase(note);
	}

	public final Phrase getValue() {
		return phrase;
	}

}
```

MTunesSymbolTable.java

```
/*
 * Created on 2003/12/07
 */
import java.util.HashMap;

/**
 * @author Hideki Sano, Huitao Sheng
 */
public class MTunesSymbolTable {

	private MTunesSymbolTable parent = null;
	private HashMap hash;

	public MTunesSymbolTable() {
		hash = new HashMap();
	}

	public final void setValue(MTunesVariable data) {
		hash.put(data.getName(), data);
	}

	public final MTunesVariable getValue(String name) {

		Object x = hash.get(name);
		MTunesSymbolTable p = getParent();

		while (x == null && p != null) {
			x = p.getValue(name);
			p = p.getParent();
		}

		return (MTunesVariable) x;

	}
```

```
	public final void setParent(MTunesSymbolTable p) {
		parent = p;
	}

	public final MTunesSymbolTable getParent() {
		return parent;
	}
}
```

MTunesVariable.java

```
/*
 * Created on 2003/12/07
 */

/**
 * @author Hideki Sano, Huitao Sheng
 */
public class MTunesVariable {

      public static final int NOTE = 1;
      public static final int SEQUENCE = 2;
      public static final int FILE = 3;
      public static final int NUM = 4;
      public static final int FUNCTION = 5;
      public static final int BOOL = 6;

      private int type;
      private String name;

      public MTunesVariable(int type, String name) {
            this.type = type;
            this.name = name;
      }

      public final String getName() {
            return name;
      }

      public final int getType() {
            return type;
      }

      public final String getTypeName() {
```

```
            String result = null;

            switch (type) {
                  case NOTE :
                        result = "Note";
                        break;
                  case SEQUENCE :
                        result = "Sequence";
                        break;
                  case FILE :
                        result = "File";
                        break;
                  case NUM :
                        result = "Number";
                        break;
                  case FUNCTION :
                        result = "Function";
                        break;
                  case BOOL :
                        result = "Bool";
                        break;
            }

            return result;
      }
}
```

MTunesVocabTokenTypes.java

```
// $ANTLR 2.7.2: "grammar.g" -> "MTunesParser.java"$

public interface MTunesVocabTokenTypes {
	int EOF = 1;
	int NULL_TREE_LOOKAHEAD = 3;
	int LOWER = 4;
	int UPPER = 5;
	int UNDER = 6;
	int ALPHA = 7;
	int DIGIT = 8;
	int PITCH = 9;
	int WS = 10;
	int NL = 11;
	int COMMENT = 12;
	int ID = 13;
	int TYPE = 14;
	int LPAREN = 15;
	int RPAREN = 16;
	int LBRACE = 17;
	int RBRACE = 18;
	int COLON = 19;
	int COMMA = 20;
	int DOT = 21;
	int MULT = 22;
	int PLUS = 23;
	int MINUS = 24;
	int DIV = 25;
	int MOD = 26;
	int SHARP = 27;
	int FLAT = 28;
	int ASGN = 29;
	int GE = 30;
	int LE = 31;
	int GT = 32;
```

```
int LT = 33;
int EQ = 34;
int NEQ = 35;
int AND = 36;
int OR = 37;
int NOT = 38;
int NUM = 39;
int NOTE = 40;
int STRING = 41;
int INSTRUMENT = 42;
int INSTRUMENT_NAME = 43;
int STATEMENT = 44;
int FORLOOP_STMT = 45;
int PRINT_STMT = 46;
int BLOCK_STMT = 47;
int FUNCT_CALL_EXPR = 48;
int FOR_EXPR = 49;
int PARAM_EXPR = 50;
int DEC_PARAM_EXPR = 51;
int FUNCT_STMT = 52;
int ID_EXPR = 53;
int MATH_TERM = 54;
int LEFT_EXPR = 55;
int RIGHT_EXPR = 56;
int NUMID_EXPR = 57;
int UMINUS = 58;
int DEC_EXPR = 59;
int DEC_ASGN = 60;
int LEFT_ASGN = 61;
int PLUS_MATH_TERM = 62;
int MINUS_MATH_TERM = 63;
int LITERAL_for = 64;
int LITERAL_foreach = 65;
int LITERAL_if = 66;
int LITERAL_else = 67;
int LITERAL_break = 68;
```

```
	int LITERAL_return = 69;
	int LITERAL_add = 70;
	int LITERAL_print = 71;
	int LITERAL_procedure = 72;
	int LITERAL_function = 73;
	int LITERAL_save = 74;
	int LITERAL_playSong = 75;
	int LITERAL_pitch = 76;
	int LITERAL_length = 77;
	int LITERAL_volume = 78;
}
```

MTunesWalker.java

```
// $ANTLR 2.7.2: "walker.g" -> "MTunesWalker.java"$

import antlr.TreeParser;
import antlr.Token;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.ANTLRException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.collections.impl.BitSet;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

import java.io.*;
import java.util.*;

import jm.JMC;
import jm.music.data.*;
import jm.util.*;
import jm.audio.*;

public class MTunesWalker
      extends antlr.TreeParser
      implements MTunesWalkerTokenTypes {

      static Score myScore = new Score("my score");
      static Part[] partArray = new Part[16];

      MTunesInterpreter mti = new MTunesInterpreter();
      public MTunesWalker() {
            tokenNames = _tokenNames;
      }
```

```
public final MTunesInterpreter prog(AST _t)
      throws RecognitionException {
      MTunesInterpreter r;

      AST prog_AST_in = (AST) _t;
      AST p = null;
      AST next = null;

      r = null;

      try { // for error handling
            AST __t2 = _t;
            p = _t == ASTNULL ? null : (AST) _t;
            match(_t, STATEMENT);
            _t = _t.getFirstChild();
            next = (AST) _t;
            if (_t == null)
                  throw new MismatchedTokenException();
            _t = _t.getNextSibling();
            _t = __t2;
            _t = _t.getNextSibling();

            MTunesVariable mtv;

            while (next != null) {
                  mtv = expr(next);
                  next = next.getNextSibling();
            }

            if (mti.isSave()) {
                  mti.saveMidiFile(myScore);
            }
            r = mti;

      } catch (RecognitionException ex) {
            reportError(ex);
```

```
                if (_t != null) {
                      _t = _t.getNextSibling();
                }
          }
          _retTree = _t;
          return r;
    }

    public final MTunesVariable expr(AST _t)
          throws RecognitionException {
          MTunesVariable r;

          AST expr_AST_in = (AST) _t;
          AST right_or = null;
          AST right_and = null;
          AST expr_dot = null;
          AST expr_id = null;
          AST dec_expr = null;
          AST inst = null;
          AST inst_name = null;
          AST next = null;
          AST left_expr = null;
          AST aNote = null;
          AST bNote = null;
          AST pure_note = null;
          AST uminus = null;
          AST num = null;
          AST idRef = null;
          AST fl = null;
          AST forbody = null;
          AST fe = null;
          AST foreachbody = null;
          AST text = null;
          AST saveFile = null;
          AST block_stmt_next = null;
          AST node = null;
```

```
            AST procID = null;
            AST procParamExpr = null;
            AST procBody = null;
            AST funcID = null;
            AST funcParamExpr = null;
            AST funcBody = null;
            AST add_inst = null;
            AST add_id = null;
            AST true_body = null;
            AST false_body = null;

            MTunesVariable a, b;
            MTunesNumber n;
            MTunesVariable[] x;
            r = null;

            try { // for error handling
                  if (_t == null)
                        _t = ASTNULL;
                  switch (_t.getType()) {
                        case OR :
                              {
                                    AST __t4 = _t;
                                    AST tmp1_AST_in = (AST) _t;
                                    match(_t, OR);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    right_or = (AST) _t;
                                    if (_t == null)
                                          throw                    new
MismatchedTokenException();
                                    _t = _t.getNextSibling();
                                    _t = __t4;
                                    _t = _t.getNextSibling();
```

```
                                    if (a.getType()
                                          != MTunesVariable.BOOL) {
                                          System.err.println(
                                                "Inconsistent  Variable
Type: "
                                                      + a.getName());
                                          System.exit(1);
                                    }
                                    r =
                                          ((MTunesBool) a).getValue()
                                                ? a
                                                : expr(right_or);

                                    break;
                              }
                        case AND :
                              {
                                    AST __t5 = _t;
                                    AST tmp2_AST_in = (AST) _t;
                                    match(_t, AND);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    right_and = (AST) _t;
                                    if (_t == null)
                                          throw                    new
MismatchedTokenException();
                                    _t = _t.getNextSibling();
                                    _t = __t5;
                                    _t = _t.getNextSibling();

                                    if (a.getType()
                                          != MTunesVariable.BOOL) {
                                          System.err.println(
                                                "Inconsistent  Variable
Type: "
```

```
                                                + a.getName());
                                    System.exit(1);
                              }
                              r =
                                    ((MTunesBool) a).getValue()
                                          ? expr(right_and)
                                          : a;

                              break;
                        }
                  case NOT :
                        {
                              AST __t6 = _t;
                              AST tmp3_AST_in = (AST) _t;
                              match(_t, NOT);
                              _t = _t.getFirstChild();
                              a = expr(_t);
                              _t = _retTree;
                              _t = __t6;
                              _t = _t.getNextSibling();

                              if (a.getType()
                                    != MTunesVariable.BOOL) {
                                    System.err.println(
                                          "Inconsistent  Variable
Type: "
                                                + a.getName());
                                    System.exit(1);
                              }
                              r = ((MTunesBool) a).not();

                              break;
                        }
                  case GE :
                        {
                              AST __t7 = _t;
```

```
                                    AST tmp4_AST_in = (AST) _t;
                                    match(_t, GE);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    b = expr(_t);
                                    _t = _retTree;
                                    _t = __t7;
                                    _t = _t.getNextSibling();

                                    if (a.getType()
                                          != MTunesVariable.NUM) {
                                          System.err.println(
                                                "Inconsistent  Variable
Type Error: "
                                                      + a.getName());
                                          System.exit(1);
                                    }
                                    if (b.getType()
                                          != MTunesVariable.NUM) {
                                          System.err.println(
                                                "Inconsistent  Variable
Type Error: "
                                                      + b.getName());
                                          System.exit(1);
                                    }
                                    boolean result =
                                          ((MTunesNumber)
a).getValue()
                                                >= ((MTunesNumber) b)
                                                      .getValue();
                                    r = new MTunesBool(null, result);

                                    break;
                              }
                        case LE :
```

```
                                {
                                      AST __t8 = _t;
                                      AST tmp5_AST_in = (AST) _t;
                                      match(_t, LE);
                                      _t = _t.getFirstChild();
                                      a = expr(_t);
                                      _t = _retTree;
                                      b = expr(_t);
                                      _t = _retTree;
                                      _t = __t8;
                                      _t = _t.getNextSibling();

                                      if (a.getType()
                                            != MTunesVariable.NUM) {
                                            System.err.println(
                                                  "Inconsistent  Variable
Type Error: "
                                                        + a.getName());
                                            System.exit(1);
                                      }
                                      if (b.getType()
                                            != MTunesVariable.NUM) {
                                            System.err.println(
                                                  "Inconsistent  Variable
Type Error: "
                                                        + b.getName());
                                            System.exit(1);
                                      }
                                      boolean result =
                                            ((MTunesNumber)
a).getValue()
                                                  <= ((MTunesNumber) b)
                                                        .getValue();
                                      r = new MTunesBool(null, result);

                                      break;
```

```
                        }
                  case GT :
                        {
                              AST __t9 = _t;
                              AST tmp6_AST_in = (AST) _t;
                              match(_t, GT);
                              _t = _t.getFirstChild();
                              a = expr(_t);
                              _t = _retTree;
                              b = expr(_t);
                              _t = _retTree;
                              _t = __t9;
                              _t = _t.getNextSibling();

                              if (a.getType()
                                    != MTunesVariable.NUM) {
                                    System.err.println(
                                          "Inconsistent  Variable
Type Error: "
                                                + a.getName());
                                    System.exit(1);
                              }
                              if (b.getType()
                                    != MTunesVariable.NUM) {
                                    System.err.println(
                                          "Inconsistent  Variable
Type Error: "
                                                + b.getName());
                                    System.exit(1);
                              }
                              boolean result =
                                    ((MTunesNumber)
a).getValue()
                                          > ((MTunesNumber) b)
                                                .getValue();
                              r = new MTunesBool(null, result);
```

```
                                    break;
                              }
                        case LT :
                              {
                                    AST __t10 = _t;
                                    AST tmp7_AST_in = (AST) _t;
                                    match(_t, LT);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    b = expr(_t);
                                    _t = _retTree;
                                    _t = __t10;
                                    _t = _t.getNextSibling();

                                    if (a.getType()
                                          != MTunesVariable.NUM) {
                                          System.err.println(
                                                "Inconsistent  Variable
Type Error: "
                                                      + a.getName());
                                          System.exit(1);
                                    }
                                    if (b.getType()
                                          != MTunesVariable.NUM) {
                                          System.err.println(
                                                "Inconsistent  Variable
Type Error: "
                                                      + b.getName());
                                          System.exit(1);
                                    }
                                    boolean result =
                                          ((MTunesNumber)
a).getValue()
                                                < ((MTunesNumber) b)
```

```
                                                .getValue();
                                    r = new MTunesBool(null, result);

                                    break;
                              }
                        case EQ :
                              {
                                    AST __t11 = _t;
                                    AST tmp8_AST_in = (AST) _t;
                                    match(_t, EQ);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    b = expr(_t);
                                    _t = _retTree;
                                    _t = __t11;
                                    _t = _t.getNextSibling();

                                    if (a.getType()
                                          != MTunesVariable.NUM) {
                                          System.err.println(
                                                "Inconsistent  Variable
Type Error: "
                                                      + a.getName());
                                          System.exit(1);
                                    }
                                    if (b.getType()
                                          != MTunesVariable.NUM) {
                                          System.err.println(
                                                "Inconsistent  Variable
Type Error: "
                                                      + b.getName());
                                          System.exit(1);
                                    }
                                    boolean result =
                                          ((MTunesNumber)
```

```
a).getValue()
                                                == ((MTunesNumber) b)
                                                      .getValue();
                                    r = new MTunesBool(null, result);

                                    break;
                              }
                        case NEQ :
                              {
                                    AST __t12 = _t;
                                    AST tmp9_AST_in = (AST) _t;
                                    match(_t, NEQ);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    b = expr(_t);
                                    _t = _retTree;
                                    _t = __t12;
                                    _t = _t.getNextSibling();

                                    if (a.getType()
                                          != MTunesVariable.NUM) {
                                          System.err.println(
                                                "Inconsistent  Variable
Type Error: "
                                                      + a.getName());
                                          System.exit(1);
                                    }
                                    if (b.getType()
                                          != MTunesVariable.NUM) {
                                          System.err.println(
                                                "Inconsistent  Variable
Type Error: "
                                                      + b.getName());
                                          System.exit(1);
                                    }
```

```
                                    boolean result =
                                          ((MTunesNumber)
a).getValue()
                                                != ((MTunesNumber) b)
                                                      .getValue();
                                    r = new MTunesBool(null, result);

                                    break;
                              }
                        case PLUS :
                              {
                                    AST __t13 = _t;
                                    AST tmp10_AST_in = (AST) _t;
                                    match(_t, PLUS);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    b = expr(_t);
                                    _t = _retTree;
                                    _t = __t13;
                                    _t = _t.getNextSibling();

                                    Phrase p = null;
                                    switch (a.getType()) {
                                          case MTunesVariable.NUM :
                                                if (b.getType()
                                                      !=
MTunesVariable.NUM) {

      System.err.println(

      "Inconsistent Variable Type Error: "
                                                                        +
a.getName()
                                                                        + ",
"
```

```
                                                                    +
b.getName());
                                                          System.exit(1);
                                                    }
                                                    double result =
                                                          ((MTunesNumber)
a)

      .getValue()
                                                                +
((MTunesNumber) b)

      .getValue();
                                                    r =
                                                          new
MTunesNumber(
                                                                null,
                                                                result);
                                                    break;
                                              case MTunesVariable.NOTE :
                                                    p = new Phrase();
                                                    if (b.getType()
                                                          ==
MTunesVariable.NOTE) {
                                                          p.add(

      ((MTunesNote) a)

      .getValue()

      .copy());
                                                          p.add(

      ((MTunesNote) b)

      .getValue()
```

```
      .copy());
                                                } else if (
                                                      b.getType()
                                                            ==
MTunesVariable

      .SEQUENCE) {
                                                      p.add(

      ((MTunesNote) a)

      .getValue()

      .copy());
                                                      p.addNoteList(
                                                            (

      (MTunesSequence) b)

      .getValue()

      .copy()

      .getNoteArray());
                                                } else {

      System.err.println(

      "Inconsistent Variable Type Error: "
                                                                        +
a.getName()
                                                                        +  ",
"
                                                                        +
b.getName());
```

```
                                                      System.exit(1);
                                                }
                                                r =
                                                      new
MTunesSequence(
                                                            null,
                                                            p);
                                                break;
                                          case  MTunesVariable.SEQUENCE
:
                                                p = new Phrase();
                                                if (b.getType()
                                                      ==
MTunesVariable.NOTE) {
                                                      p.addNoteList(
                                                            (

      (MTunesSequence) a)

      .getValue()

      .copy()

      .getNoteArray());
                                                      p.add(

      ((MTunesNote) b)

      .getValue()

      .copy());
                                                } else if (
                                                      b.getType()
                                                            ==
MTunesVariable
```

```
      .SEQUENCE) {
                                                        p.addNoteList(
                                                              (

      (MTunesSequence) a)

      .getValue()

      .copy()

      .getNoteArray());
                                                        p.addNoteList(
                                                              (

      (MTunesSequence) b)

      .getValue()

      .copy()

      .getNoteArray());
                                                  } else {

      System.err.println(

      "Inconsistent Variable Type Error: "
                                                                        +
a.getName()
                                                                        +  ",
"
                                                                        +
b.getName());
                                                        System.exit(1);
                                                  }
                                                  r =
                                                        new
```

```
MTunesSequence(
                                                        null,
                                                        p);
                                              break;
                                        }

                                        break;
                                  }
                            case MINUS :
                                  {
                                        AST __t14 = _t;
                                        AST tmp11_AST_in = (AST) _t;
                                        match(_t, MINUS);
                                        _t = _t.getFirstChild();
                                        a = expr(_t);
                                        _t = _retTree;
                                        b = expr(_t);
                                        _t = _retTree;
                                        _t = __t14;
                                        _t = _t.getNextSibling();

                                        if (a.getType()
                                              != MTunesVariable.NUM) {
                                              System.err.println(
                                                    "Inconsistent  Variable
Type Error: "
                                                          + a.getName());
                                              System.exit(1);
                                        }
                                        if (b.getType()
                                              != MTunesVariable.NUM) {
                                              System.err.println(
                                                    "Inconsistent  Variable
Type Error: "
                                                          + b.getName());
                                              System.exit(1);
```

```
                                    }
                                    double result =
                                          ((MTunesNumber)
a).getValue()
                                                - ((MTunesNumber) b)
                                                      .getValue();
                                    r    =    new    MTunesNumber(null,
result);

                                    break;
                              }
                        case MULT :
                              {
                                    AST __t15 = _t;
                                    AST tmp12_AST_in = (AST) _t;
                                    match(_t, MULT);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    b = expr(_t);
                                    _t = _retTree;
                                    _t = __t15;
                                    _t = _t.getNextSibling();

                                    if (a.getType()
                                          != MTunesVariable.NUM) {
                                          System.err.println(
                                                "Inconsistent  Variable
Type Error: "
                                                      + a.getName());
                                          System.exit(1);
                                    }
                                    if (b.getType()
                                          != MTunesVariable.NUM) {
                                          System.err.println(
                                                "Inconsistent  Variable
```

```
Type Error: "
                                                + b.getName());
                                          System.exit(1);
                                    }
                                    double result =
                                          ((MTunesNumber)
a).getValue()
                                                * ((MTunesNumber) b)
                                                      .getValue();
                                    r    =    new    MTunesNumber(null,
result);

                                    break;
                              }
                        case DIV :
                              {
                                    AST __t16 = _t;
                                    AST tmp13_AST_in = (AST) _t;
                                    match(_t, DIV);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    b = expr(_t);
                                    _t = _retTree;
                                    _t = __t16;
                                    _t = _t.getNextSibling();

                                    if (a.getType()
                                          != MTunesVariable.NUM) {
                                          System.err.println(
                                                "Inconsistent  Variable
Type Error: "
                                                      + a.getName());
                                          System.exit(1);
                                    }
                                    if (b.getType()
```

```
                                        != MTunesVariable.NUM) {
                                        System.err.println(
                                              "Inconsistent  Variable
Type Error: "
                                                    + b.getName());
                                        System.exit(1);
                                  }
                                  double result =
                                        ((MTunesNumber)
a).getValue()
                                              / ((MTunesNumber) b)
                                                    .getValue();
                                  r    =    new    MTunesNumber(null,
result);

                                  break;
                            }
                      case MOD :
                            {
                                  AST __t17 = _t;
                                  AST tmp14_AST_in = (AST) _t;
                                  match(_t, MOD);
                                  _t = _t.getFirstChild();
                                  a = expr(_t);
                                  _t = _retTree;
                                  b = expr(_t);
                                  _t = _retTree;
                                  _t = __t17;
                                  _t = _t.getNextSibling();

                                  if (a.getType()
                                        != MTunesVariable.NUM) {
                                        System.err.println(
                                              "Inconsistent  Variable
Type Error: "
                                                    + a.getName());
```

```
                                    System.exit(1);
                              }
                              if (b.getType()
                                    != MTunesVariable.NUM) {
                                    System.err.println(
                                          "Inconsistent  Variable
Type Error: "
                                                + b.getName());
                                    System.exit(1);
                              }
                              int i1 =
                                    (int) (((MTunesNumber) a)
                                          .getValue());
                              int i2 =
                                    (int) (((MTunesNumber) b)
                                          .getValue());
                              double result = i1 % i2;
                              r    =    new    MTunesNumber(null,
result);

                              break;
                        }
                  case MATH_TERM :
                        {
                              AST __t18 = _t;
                              AST tmp15_AST_in = (AST) _t;
                              match(_t, MATH_TERM);
                              _t = _t.getFirstChild();
                              a = expr(_t);
                              _t = _retTree;
                              _t = __t18;
                              _t = _t.getNextSibling();
                              r = a;
                              break;
                        }
                  case PLUS_MATH_TERM :
```

```
                        {
                              AST __t19 = _t;
                              AST tmp16_AST_in = (AST) _t;
                              match(_t, PLUS_MATH_TERM);
                              _t = _t.getFirstChild();
                              a = expr(_t);
                              _t = _retTree;
                              _t = __t19;
                              _t = _t.getNextSibling();
                              r = a;
                              break;
                        }
                  case MINUS_MATH_TERM :
                        {
                              AST __t20 = _t;
                              AST tmp17_AST_in = (AST) _t;
                              match(_t, MINUS_MATH_TERM);
                              _t = _t.getFirstChild();
                              a = expr(_t);
                              _t = _retTree;
                              _t = __t20;
                              _t = _t.getNextSibling();

                              r =
                                    new MTunesNumber(
                                          null,
                                          ((MTunesNumber) a)
                                                .getValue()
                                                * (-1));

                              break;
                        }
                  case ID_EXPR :
                        {
                              AST __t21 = _t;
                              AST tmp18_AST_in = (AST) _t;
```

```
                                    match(_t, ID_EXPR);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    {
                                          _loop23 : do {
                                                if (_t == null)
                                                      _t = ASTNULL;
                                                if ((_t.getType()
                                                      == DOT)) {
                                                      expr_dot = (AST)
_t;
                                                      match(_t, DOT);
                                                      _t =

      _t.getNextSibling();
                                                      expr_id = (AST)
_t;
                                                      if (_t == null)
                                                            throw new
MismatchedTokenException();
                                                      _t =

      _t.getNextSibling();
                                                } else {
                                                      break _loop23;
                                                }

                                          } while (true);
                                    }
                                    _t = __t21;
                                    _t = _t.getNextSibling();

                                    if (expr_dot == null) {
                                          r = a;
                                    } else {
```

```
                                        if (expr_id
                                              .getText()
                                              .compareTo("pitch")
                                              == 0) {
                                              double d =
                                                    ((MTunesNote) a)

      .getValue()

      .getPitch();
                                              r =
                                                    new
MTunesNumber(
                                                          null,
                                                          d);
                                        } else if (
                                              expr_id
                                                    .getText()
                                                    .compareTo(
                                                    "length")
                                                    == 0) {
                                              double d =
                                                    ((MTunesNote) a)

      .getValue()

      .getDuration();
                                              r =
                                                    new
MTunesNumber(
                                                          null,
                                                          d);
                                        } else if (
                                              expr_id
                                                    .getText()
                                                    .compareTo(
```

```
                                                      "volume")
                                                      == 0) {
                                                double d =
                                                      ((MTunesNote) a)

      .getValue()

      .getDynamic();
                                                r =
                                                      new
MTunesNumber(
                                                            null,
                                                            d);
                                          }
                                    }

                                    break;
                              }
                        case FUNCT_CALL_EXPR :
                              {
                                    AST __t24 = _t;
                                    AST tmp19_AST_in = (AST) _t;
                                    match(_t, FUNCT_CALL_EXPR);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    x = mexpr(_t);
                                    _t = _retTree;
                                    _t = __t24;
                                    _t = _t.getNextSibling();

                                    r = mti.invokeFunct(this, a, x);

                                    break;
                              }
                        case LITERAL_return :
```

```
                              {
                                    AST __t25 = _t;
                                    AST tmp20_AST_in = (AST) _t;
                                    match(_t, LITERAL_return);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    _t = __t25;
                                    _t = _t.getNextSibling();
                                    r = a;
                                    break;
                              }
                        case DEC_EXPR :
                              {
                                    AST __t26 = _t;
                                    dec_expr =
                                          _t == ASTNULL ? null : (AST)
_t;
                                    match(_t, DEC_EXPR);
                                    _t = _t.getFirstChild();
                                    _t = __t26;
                                    _t = _t.getNextSibling();

                                    AST dec_type =
                                          dec_expr.getFirstChild();
                                    AST dec_id =
                                          dec_type.getNextSibling();
                                    r =
                                          mti.createVariable(
                                                dec_type.getText(),
                                                dec_id.getText());

                                    break;
                              }
                        case ASGN :
                              {
```

```
                                    AST __t27 = _t;
                                    AST tmp21_AST_in = (AST) _t;
                                    match(_t, ASGN);
                                    _t = _t.getFirstChild();
                                    inst = (AST) _t;
                                    match(_t, INSTRUMENT);
                                    _t = _t.getNextSibling();
                                    inst_name = (AST) _t;
                                    match(_t, INSTRUMENT_NAME);
                                    _t = _t.getNextSibling();
                                    _t = __t27;
                                    _t = _t.getNextSibling();

                                    int instrumentNumber =
                                          Integer.parseInt(

      inst.getText().substring(
                                                          1))
                                                - 1;

                                    if (partArray[instrumentNumber]
                                          == null) {
                                          partArray[instrumentNumber]
=
                                                new Part(
                                                      mti

      .getInstrumentNumber(
                                                                inst_name

      .getText()),

      instrumentNumber);
                                    } else {
                                          partArray[instrumentNumber]
                                                .setInstrument(
```

```
mti.getInstrumentNumber(

inst_name.getText()));
                                }

                                myScore.addPart(

partArray[instrumentNumber]);

                                break;
                          }
                    case DEC_ASGN :
                          {
                                AST __t28 = _t;
                                AST tmp22_AST_in = (AST) _t;
                                match(_t, DEC_ASGN);
                                _t = _t.getFirstChild();
                                a = expr(_t);
                                _t = _retTree;
                                {
                                      if (_t == null)
                                            _t = ASTNULL;
                                      switch (_t.getType()) {
                                            case LOWER :
                                            case UPPER :
                                            case UNDER :
                                            case ALPHA :
                                            case DIGIT :
                                            case PITCH :
                                            case WS :
                                            case NL :
                                            case COMMENT :
                                            case ID :
                                            case TYPE :
                                            case LPAREN :
```

```
                                        case RPAREN :
                                        case LBRACE :
                                        case RBRACE :
                                        case COLON :
                                        case COMMA :
                                        case DOT :
                                        case MULT :
                                        case PLUS :
                                        case MINUS :
                                        case DIV :
                                        case MOD :
                                        case SHARP :
                                        case FLAT :
                                        case ASGN :
                                        case GE :
                                        case LE :
                                        case GT :
                                        case LT :
                                        case EQ :
                                        case NEQ :
                                        case AND :
                                        case OR :
                                        case NOT :
                                        case NUM :
                                        case NOTE :
                                        case STRING :
                                        case INSTRUMENT :
                                        case INSTRUMENT_NAME :
                                        case STATEMENT :
                                        case FORLOOP_STMT :
                                        case PRINT_STMT :
                                        case BLOCK_STMT :
                                        case FUNCT_CALL_EXPR :
                                        case FOR_EXPR :
                                        case PARAM_EXPR :
                                        case DEC_PARAM_EXPR :
```

```
                                                    case FUNCT_STMT :
                                                    case ID_EXPR :
                                                    case MATH_TERM :
                                                    case LEFT_EXPR :
                                                    case RIGHT_EXPR :
                                                    case NUMID_EXPR :
                                                    case UMINUS :
                                                    case DEC_EXPR :
                                                    case DEC_ASGN :
                                                    case LEFT_ASGN :
                                                    case PLUS_MATH_TERM :
                                                    case MINUS_MATH_TERM :
                                                    case LITERAL_for :
                                                    case LITERAL_foreach :
                                                    case LITERAL_if :
                                                    case LITERAL_else :
                                                    case LITERAL_break :
                                                    case LITERAL_return :
                                                    case LITERAL_add :
                                                    case LITERAL_print :
                                                    case  LITERAL_procedure
:
                                                    case   LITERAL_function
:
                                                    case LITERAL_save :
                                                    case   LITERAL_playSong
:
                                                    case LITERAL_pitch :
                                                    case LITERAL_length :
                                                    case LITERAL_volume :
                                                          {
                                                                next      =
(AST) _t;
                                                                if  (_t  ==
null)
```

```
      throw new MismatchedTokenException();
                                                              _t =
                                                                    _t

      .getNextSibling();
                                                              break;
                                                        }
                                                  case 3 :
                                                        {
                                                              break;
                                                        }
                                                  default :
                                                        {
                                                              throw   new
NoViableAltException(_t);
                                                        }
                                            }
                                      }
                                      _t = __t28;
                                      _t = _t.getNextSibling();

                                      if (next != null) {

                                            b = expr(next);

                                            switch (a.getType()) {

                                                  case
MTunesVariable.NUM :
                                                        if (b.getType()
                                                              !=
MTunesVariable

      .NUM) {

      System.err.println(
```

```
      "Inconsistent Variable Type Error: "
      + a.getName()
      + ", "
      + b
            .getName());
      System.exit(1);
                                                      }
                                                      (
                                                            (
      MTunesNumber) a)
      .setValue(
      ((MTunesNumber) b)
      .getValue());
                                                      break;

                                                case
MTunesVariable.NOTE :
                                                      if (b.getType()
                                                            !=
MTunesVariable
      .NOTE) {
      System.err.println(
      "Inconsistent Variable Type Error: "
```

```
      + a.getName()
      + ", "
      + b
            .getName());
      System.exit(1);
                                                      }
                                                      (
                                                            (
      MTunesNote) a)
      .setValue(
      ((MTunesNote) b)
      .getValue());
                                                      break;
                                                case MTunesVariable
                                                      .SEQUENCE :
                                                      switch
(b.getType()) {
                                                            case
MTunesVariable
      .NOTE :
                                                                  (
      (
```

```
            MTunesSequence) a)

                  .setValue(

      (

            (MTunesNote) b)

            .getValue());

      break;
                                                            case
MTunesVariable

      .SEQUENCE :
                                                                  (

      (

            MTunesSequence) a)

                  .setValue(

      (

            (MTunesSequence) b)

            .getValue());

      break;
                                                            default :

      System

      .err
```

```
      .println(
      "Inconsistent Variable Type Error: "
            + a
                  .getName()
            + ", "
            + b
                  .getName());
      System.exit(1);
      break;
                                                      }
                                                      break;
                                                }
                                    }
                                    r = a;
                                    break;
                              }
                        case LEFT_ASGN :
                              {
                                    AST __t30 = _t;
                                    AST tmp23_AST_in = (AST) _t;
                                    match(_t, LEFT_ASGN);
                                    _t = _t.getFirstChild();
                                    left_expr = (AST) _t;
                                    match(_t, LEFT_EXPR);
```

```
                                    _t = _t.getNextSibling();
                                    a = expr(_t);
                                    _t = _retTree;
                                    _t = __t30;
                                    _t = _t.getNextSibling();

                                    AST left =
                                          left_expr
                                                .getFirstChild()
                                                .getFirstChild();
                                    String name = left.getText();
                                    AST dot = left.getNextSibling();

                                    if (a == null) {
                                          /*
                                           *  For  example,  right  hand
side
                                           * is a procedure.
                                           */
                                          System.err.println(
                                                "Invalid    Assignment
Statement Error: "
                                                      + name);
                                          System.err.println(
                                                "Procedure     doesn't
return a value");
                                          System.exit(1);
                                    }

                                    if (dot == null) {
                                          mti.setVariable(name, a);
                                    } else {
                                          String attr =
                                                dot

      .getNextSibling()
```

```
                                                .getText();
                                    double d =
                                          ((MTunesNumber) a)
                                                .getValue();
                                    mti.setNoteAttribute(
                                          name,
                                          attr,
                                          d);
                              }

                              break;
                        }
                  case RIGHT_EXPR :
                        {
                              AST __t31 = _t;
                              AST tmp24_AST_in = (AST) _t;
                              match(_t, RIGHT_EXPR);
                              _t = _t.getFirstChild();
                              aNote = (AST) _t;
                              match(_t, NOTE);
                              _t = _t.getNextSibling();

                              Note r1 = null;
                              Phrase r2 = null;
                              int singleNote = 1;

                              if (aNote.getNextSibling()
                                    == null) {
                                    r1 =
                                          mti.intpNote(

      aNote.getText());
                              } else {
                                    singleNote = 2;
                                    r2 = new Phrase();
                                    r2.addNote(
```

```
                                                mti.intpNote(

      aNote.getText()));
                                    }

                                    {
                                          _loop33 : do {
                                                if (_t == null)
                                                      _t = ASTNULL;
                                                if ((_t.getType()
                                                      == NOTE)) {
                                                      bNote   =   (AST)
_t;
                                                      match(_t, NOTE);
                                                      _t =

      _t.getNextSibling();

                                                      r2.addNote(

      mti.intpNote(

      bNote

      .getText()));

                                                } else {
                                                      break _loop33;
                                                }

                                          } while (true);
                                    }
                                    _t = __t31;
                                    _t = _t.getNextSibling();

                                    if (singleNote == 1) {
```

```
                                    r =
                                          new MTunesNote(
                                                "unknown",
                                                r1);
                                    } else {
                                          r =
                                          new MTunesSequence(
                                                "unknown",
                                                r2);
                                    }

                                    break;
                              }
                        case NOTE :
                              {
                                    pure_note = (AST) _t;
                                    match(_t, NOTE);
                                    _t = _t.getNextSibling();

                                    Note result =

mti.intpNote(aNote.getText());
                                    r =
                                          new MTunesNote(
                                                "unknown",
                                                result);

                                    break;
                              }
                        case UMINUS :
                              {
                                    AST __t34 = _t;
                                    AST tmp25_AST_in = (AST) _t;
                                    match(_t, UMINUS);
                                    _t = _t.getFirstChild();
                                    uminus = (AST) _t;
```

```
                                    match(_t, NUM);
                                    _t = _t.getNextSibling();
                                    _t = __t34;
                                    _t = _t.getNextSibling();
                                    r =
                                          new MTunesNumber(
                                                null,
                                                Double.parseDouble(

      uminus.getText())
                                                                * (-1));
                                    break;
                              }
                        case NUM :
                              {
                                    num = (AST) _t;
                                    match(_t, NUM);
                                    _t = _t.getNextSibling();

                                    r = mti.getNumber(num.getText());

                                    break;
                              }
                        case ID :
                              {
                                    idRef = (AST) _t;
                                    match(_t, ID);
                                    _t = _t.getNextSibling();

                                    r =
                                          mti.getVariable(
                                                idRef.getText());
                                    if (r == null) {
                                          System.err.println(
                                                "Undefined       Symbol
Error: "
```

```
                                                        +
idRef.getText());
                                          System.exit(1);
                                    }

                                    break;
                              }
                        case LITERAL_for :
                              {
                                    AST __t35 = _t;
                                    AST tmp26_AST_in = (AST) _t;
                                    match(_t, LITERAL_for);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    b = expr(_t);
                                    _t = _retTree;
                                    _t = __t35;
                                    _t = _t.getNextSibling();
                                    break;
                              }
                        case FOR_EXPR :
                              {
                                    AST __t36 = _t;
                                    AST tmp27_AST_in = (AST) _t;
                                    match(_t, FOR_EXPR);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    b = expr(_t);
                                    _t = _retTree;
                                    n = loop_counter(_t);
                                    _t = _retTree;
                                    _t = __t36;
                                    _t = _t.getNextSibling();
```

```
                                    mti.forloopInit(
                                          ((MTunesNumber)
a).getValue(),
                                          ((MTunesNumber)
b).getValue(),
                                          n);

                                    break;
                              }
                        case FORLOOP_STMT :
                              {
                                    AST __t37 = _t;
                                    fl =
                                          _t == ASTNULL ? null : (AST)
_t;
                                    match(_t, FORLOOP_STMT);
                                    _t = _t.getFirstChild();
                                    forbody = (AST) _t;
                                    if (_t == null)
                                          throw                    new
MismatchedTokenException();
                                    _t = _t.getNextSibling();
                                    _t = __t37;
                                    _t = _t.getNextSibling();

                                    while (mti.forloopCanProceed()) {

                                          AST stmt = forbody;
                                          while (stmt != null) {
                                                r = expr(stmt);
                                                if (mti

      .isReturningFromBreak()) {
                                                      break;
                                                }
                                                stmt =
```

```
      stmt.getNextSibling();
                                          }
                                          if (mti

      .isReturningFromBreak()) {
                                                break;
                                          }
                                          mti.forloopNext();

                                    }
                                    mti.completeReturnFromBreak();
                                    mti.forloopEnd();

                                    break;
                              }
                        case LITERAL_foreach :
                              {
                                    AST __t38 = _t;
                                    fe =
                                          _t == ASTNULL ? null : (AST)
_t;
                                    match(_t, LITERAL_foreach);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    b = expr(_t);
                                    _t = _retTree;
                                    foreachbody = (AST) _t;
                                    if (_t == null)
                                          throw                    new
MismatchedTokenException();
                                    _t = _t.getNextSibling();
                                    _t = __t38;
                                    _t = _t.getNextSibling();
```

```
                                    mti.foreachInit(
                                          (MTunesSequence) a,
                                          (MTunesNote) b);

                                    while (mti.foreachCanProceed()) {

                                          AST stmt =

      foreachbody.getFirstChild();
                                          while (stmt != null) {
                                                r = expr(stmt);
                                                if (mti

      .isReturningFromBreak()) {
                                                      break;
                                                }
                                                stmt =

      stmt.getNextSibling();
                                          }
                                          if (mti

      .isReturningFromBreak()) {
                                                break;
                                          }
                                          mti.foreachNext();

                                    }
                                    mti.completeReturnFromBreak();
                                    mti.foreachEnd();

                                    break;
                              }
                        case PRINT_STMT :
                              {
                                    AST __t39 = _t;
```

```
                                    AST tmp28_AST_in = (AST) _t;
                                    match(_t, PRINT_STMT);
                                    _t = _t.getFirstChild();
                                    text = (AST) _t;
                                    match(_t, STRING);
                                    _t = _t.getNextSibling();
                                    _t = __t39;
                                    _t = _t.getNextSibling();


      System.out.println(text.getText());

                                    break;
                              }
                        case LITERAL_save :
                              {
                                    AST __t40 = _t;
                                    AST tmp29_AST_in = (AST) _t;
                                    match(_t, LITERAL_save);
                                    _t = _t.getFirstChild();
                                    saveFile = (AST) _t;
                                    match(_t, STRING);
                                    _t = _t.getNextSibling();
                                    _t = __t40;
                                    _t = _t.getNextSibling();


      mti.enableSave(saveFile.getText());

                                    break;
                              }
                        case BLOCK_STMT :
                              {
                                    AST __t41 = _t;
                                    AST tmp30_AST_in = (AST) _t;
                                    match(_t, BLOCK_STMT);
```

```
                                    _t = _t.getFirstChild();
                                    block_stmt_next = (AST) _t;
                                    if (_t == null)
                                          throw                    new
MismatchedTokenException();
                                    _t = _t.getNextSibling();
                                    _t = __t41;
                                    _t = _t.getNextSibling();

                                    while (block_stmt_next != null) {
                                          r = expr(block_stmt_next);
                                          block_stmt_next =
                                                block_stmt_next

      .getNextSibling();
                                    }

                                    break;
                              }
                        case STATEMENT :
                              {
                                    AST __t42 = _t;
                                    AST tmp31_AST_in = (AST) _t;
                                    match(_t, STATEMENT);
                                    _t = _t.getFirstChild();
                                    node = (AST) _t;
                                    if (_t == null)
                                          throw                    new
MismatchedTokenException();
                                    _t = _t.getNextSibling();
                                    _t = __t42;
                                    _t = _t.getNextSibling();

                                    while (node != null) {
                                          r = expr(node);
                                          node                      =
```

```
node.getNextSibling();
                                    }

                                    break;
                              }
                        case LITERAL_procedure :
                              {
                                    AST __t43 = _t;
                                    AST tmp32_AST_in = (AST) _t;
                                    match(_t, LITERAL_procedure);
                                    _t = _t.getFirstChild();
                                    procID = (AST) _t;
                                    match(_t, ID);
                                    _t = _t.getNextSibling();
                                    procParamExpr = (AST) _t;
                                    match(_t, DEC_PARAM_EXPR);
                                    _t = _t.getNextSibling();
                                    procBody = (AST) _t;
                                    match(_t, BLOCK_STMT);
                                    _t = _t.getNextSibling();
                                    _t = __t43;
                                    _t = _t.getNextSibling();

                                    MTunesVariable mtv =
                                          mti.getVariable(
                                                procID.getText());
                                    if (mtv != null) {
                                          System.err.println(
                                                "Duplicate     Variable
Definition Error: "
                                                      +
procID.getText());
                                          System.exit(1);
                                    }

                                    MTunesVariable[] argList =
```

```
                                        new
MTunesVariable[procParamExpr

      .getNumberOfChildren()];
                                  AST decExpr =

      procParamExpr.getFirstChild();
                                  // DEC_EXPR
                                  int i = 0;
                                  while (decExpr != null) {
                                        AST argType =

      decExpr.getFirstChild();
                                        AST argName =

      argType.getNextSibling();
                                        argList[i++] =
                                              mti.createVariable(

      argType.getText(),

      argName.getText(),
                                                          false);
                                        decExpr =

      decExpr.getNextSibling();
                                        // next DEC_EXPR
                                  }
                                  MTunesFunction mtf =
                                        (
                                              MTunesFunction) mti
                                                    .createVariable(
                                              "Function",
                                              procID.getText());
                                  mtf.setArgs(argList);
                                  mtf.setBody(
```

```
                                        procBody.getFirstChild());

                                  break;
                            }
                      case LITERAL_function :
                            {
                                  AST __t44 = _t;
                                  AST tmp33_AST_in = (AST) _t;
                                  match(_t, LITERAL_function);
                                  _t = _t.getFirstChild();
                                  funcID = (AST) _t;
                                  match(_t, ID);
                                  _t = _t.getNextSibling();
                                  funcParamExpr = (AST) _t;
                                  match(_t, DEC_PARAM_EXPR);
                                  _t = _t.getNextSibling();
                                  funcBody = (AST) _t;
                                  match(_t, FUNCT_STMT);
                                  _t = _t.getNextSibling();
                                  _t = __t44;
                                  _t = _t.getNextSibling();

                                  MTunesVariable mtv =
                                        mti.getVariable(
                                              funcID.getText());
                                  if (mtv != null) {
                                        System.err.println(
                                              "Duplicate     Variable
Definition Error: "
                                                    +
funcID.getText());
                                        System.exit(1);
                                  }

                                  MTunesVariable[] argList =
                                        new
```

```
MTunesVariable[funcParamExpr

      .getNumberOfChildren()];
                                    AST decExpr =

      funcParamExpr.getFirstChild();
                                    // DEC_EXPR
                                    int i = 0;
                                    while (decExpr != null) {
                                          AST argType =

      decExpr.getFirstChild();
                                          AST argName =

      argType.getNextSibling();
                                          argList[i++] =
                                                mti.createVariable(

      argType.getText(),

      argName.getText(),
                                                            false);
                                          decExpr =

      decExpr.getNextSibling();
                                          // next DEC_EXPR
                                    }
                                    MTunesFunction mtf =
                                          (
                                                MTunesFunction) mti
                                                      .createVariable(
                                                "Function",
                                                funcID.getText());
                                    mtf.setArgs(argList);
                                    mtf.setBody(
                                          funcBody.getFirstChild());
```

```
                        break;
                }
            case LITERAL_playSong :
                {
                        AST __t45 = _t;
                        AST tmp34_AST_in = (AST) _t;
                        match(_t, LITERAL_playSong);
                        _t = _t.getFirstChild();
                        _t = __t45;
                        _t = _t.getNextSibling();

                        if (mti.isSave()) {
                              mti.saveMidiFile(myScore);
                        }

                        mti.checkScore(myScore);

                        View.notate(myScore);
                        Play.midi(myScore);

                        break;
                }
            case LITERAL_add :
                {
                        AST __t46 = _t;
                        AST tmp35_AST_in = (AST) _t;
                        match(_t, LITERAL_add);
                        _t = _t.getFirstChild();
                        add_inst = (AST) _t;
                        match(_t, INSTRUMENT);
                        _t = _t.getNextSibling();
                        add_id = (AST) _t;
                        match(_t, ID);
                        _t = _t.getNextSibling();
                        _t = __t46;
```

```
                                    _t = _t.getNextSibling();

                                    int instNum =
                                          Integer.parseInt(
                                                add_inst
                                                      .getText()
                                                      .substring(
                                                      1))
                                                - 1;
                                    Part p = partArray[instNum];

                                    if (p == null) {
                                          System.err.println(
                                                "Uninitialized
Instrument Error: "
                                                      +    (instNum    +
1));
                                          System.exit(1);
                                    }

                                    do {

                                          MTunesVariable mtv =
                                                mti.getVariable(

      add_id.getText());
                                          if (mtv == null) {
                                                System.err.println(
                                                      "Undefined
Symbol Error: "
                                                            +
add_id.getText());
                                                System.exit(1);
                                          }

                                          switch (mtv.getType()) {
```

```
                                                    case
MTunesVariable.NOTE :
                                                          double endTime =

      p.getEndTime();
                                                          p.addNote(

      ((MTunesNote) mtv)

      .getValue(),
                                                                endTime);
                                                          break;
                                                    case MTunesVariable
                                                          .SEQUENCE :
                                                          p.appendPhrase(
                                                                (

      (MTunesSequence) mtv)

      .getValue());
                                                          break;
                                                    default :

      System.err.println(

      "Inconsistent Variable Type Error: "
                                                                      +
mtv.getName());
                                                          System.exit(1);
                                                          break;
                                               }

                                               add_id =

      add_id.getNextSibling();
```

```
                                    } while (add_id != null);

                                    break;
                              }
                        case LITERAL_if :
                              {
                                    AST __t47 = _t;
                                    AST tmp36_AST_in = (AST) _t;
                                    match(_t, LITERAL_if);
                                    _t = _t.getFirstChild();
                                    a = expr(_t);
                                    _t = _retTree;
                                    true_body = (AST) _t;
                                    match(_t, BLOCK_STMT);
                                    _t = _t.getNextSibling();
                                    {
                                          if (_t == null)
                                                _t = ASTNULL;
                                          switch (_t.getType()) {
                                                case BLOCK_STMT :
                                                      {
                                                            false_body
=

      (AST) _t;
                                                            match(
                                                                  _t,

      BLOCK_STMT);
                                                            _t =
                                                                  _t

      .getNextSibling();
                                                            break;
                                                      }
                                                case 3 :
```

```
                                                {
                                                      break;
                                                }
                                          default :
                                                {
                                                      throw   new
NoViableAltException(_t);
                                                }
                                    }
                              }
                              _t = __t47;
                              _t = _t.getNextSibling();

                              if (a.getType()
                                    != MTunesVariable.BOOL) {
                                    System.err.println(
                                          "Invalid       Variable
Type: "
                                                + a.getName());
                                    System.exit(1);
                              }

                              if (((MTunesBool) a).getValue()) {
                                    AST stmt =

      true_body.getFirstChild();
                                    while (stmt != null) {
                                          r = expr(stmt);
                                          stmt =

      stmt.getNextSibling();
                                    }
                              } else if (false_body != null) {
                                    AST stmt =

      false_body.getFirstChild();
```

```
                                        while (stmt != null) {
                                              r = expr(stmt);
                                              stmt =

      stmt.getNextSibling();
                                        }
                                  }

                                  break;
                            }
                      case LITERAL_break :
                            {
                                  AST __t49 = _t;
                                  AST tmp37_AST_in = (AST) _t;
                                  match(_t, LITERAL_break);
                                  _t = _t.getFirstChild();
                                  _t = __t49;
                                  _t = _t.getNextSibling();

                                  mti.startReturnFromBreak();

                                  break;
                            }
                      default :
                            {
                                  throw                          new
NoViableAltException(_t);
                            }
                  }
            } catch (RecognitionException ex) {
                  reportError(ex);
                  if (_t != null) {
                        _t = _t.getNextSibling();
                  }
            }
            _retTree = _t;
```

```
        return r;
    }

    public final MTunesVariable[] mexpr(AST _t)
        throws RecognitionException {
        MTunesVariable[] rv;

        AST mexpr_AST_in = (AST) _t;

        MTunesVariable a;
        rv = null;
        Vector v;

        try { // for error handling
            AST __t52 = _t;
            AST tmp38_AST_in = (AST) _t;
            match(_t, PARAM_EXPR);
            _t = _t.getFirstChild();
            v = new Vector();
            {
                _loop54 : do {
                    if (_t == null)
                        _t = ASTNULL;
                    if ((_tokenSet_0
                        .member(_t.getType()))) {
                        a = expr(_t);
                        _t = _retTree;
                        v.add(a);
                    } else {
                        break _loop54;
                    }

                } while (true);
            }
            _t = __t52;
            _t = _t.getNextSibling();
```

```
                MTunesVariable[] x =
                        new MTunesVariable[v.size()];
                for (int i = 0; i < x.length; i++) {
                        x[i] = (MTunesVariable) (v.elementAt(i));
                }

                rv = x;

        } catch (RecognitionException ex) {
                reportError(ex);
                if (_t != null) {
                        _t = _t.getNextSibling();
                }
        }
        _retTree = _t;
        return rv;
}

public final MTunesNumber loop_counter(AST _t)
        throws RecognitionException {
        MTunesNumber r;

        AST loop_counter_AST_in = (AST) _t;
        AST id = null;

        r = null;

        try { // for error handling
                id = (AST) _t;
                match(_t, ID);
                _t = _t.getNextSibling();

                r = new MTunesNumber(id.getText());

        } catch (RecognitionException ex) {
```

```
                reportError(ex);
                if (_t != null) {
                      _t = _t.getNextSibling();
                }
          }
          _retTree = _t;
          return r;
    }

    public static final String[] _tokenNames =
          {
                "<0>",
                "EOF",
                "<2>",
                "NULL_TREE_LOOKAHEAD",
                "LOWER",
                "UPPER",
                "UNDER",
                "ALPHA",
                "DIGIT",
                "PITCH",
                "WS",
                "NL",
                "COMMENT",
                "ID",
                "TYPE",
                "LPAREN",
                "RPAREN",
                "LBRACE",
                "RBRACE",
                "COLON",
                "COMMA",
                "DOT",
                "MULT",
                "PLUS",
                "MINUS",
```

```
                    "DIV",
                    "MOD",
                    "SHARP",
                    "FLAT",
                    "ASGN",
                    "GE",
                    "LE",
                    "GT",
                    "LT",
                    "EQ",
                    "NEQ",
                    "AND",
                    "OR",
                    "NOT",
                    "NUM",
                    "NOTE",
                    "STRING",
                    "INSTRUMENT",
                    "INSTRUMENT_NAME",
                    "STATEMENT",
                    "FORLOOP_STMT",
                    "PRINT_STMT",
                    "BLOCK_STMT",
                    "FUNCT_CALL_EXPR",
                    "FOR_EXPR",
                    "PARAM_EXPR",
                    "DEC_PARAM_EXPR",
                    "FUNCT_STMT",
                    "ID_EXPR",
                    "MATH_TERM",
                    "LEFT_EXPR",
                    "RIGHT_EXPR",
                    "NUMID_EXPR",
                    "UMINUS",
                    "DEC_EXPR",
                    "DEC_ASGN",
```

```
                  "LEFT_ASGN",
                  "PLUS_MATH_TERM",
                  "MINUS_MATH_TERM",
                  "\"for\"",
                  "\"foreach\"",
                  "\"if\"",
                  "\"else\"",
                  "\"break\"",
                  "\"return\"",
                  "\"add\"",
                  "\"print\"",
                  "\"procedure\"",
                  "\"function\"",
                  "\"save\"",
                  "\"playSong\"",
                  "\"pitch\"",
                  "\"length\"",
                  "\"volume\"" };

      private static final long[] mk_tokenSet_0() {
            long[] data =
                  { -188040678012346368L, 3959L, 0L, 0L };
            return data;
      }
      public static final BitSet _tokenSet_0 =
            new BitSet(mk_tokenSet_0());
}
```

MTunesWalkerTokenTypes.java

```
// $ANTLR 2.7.2: "walker.g" -> "MTunesWalker.java"$

public interface MTunesWalkerTokenTypes {
	int EOF = 1;
	int NULL_TREE_LOOKAHEAD = 3;
	int LOWER = 4;
	int UPPER = 5;
	int UNDER = 6;
	int ALPHA = 7;
	int DIGIT = 8;
	int PITCH = 9;
	int WS = 10;
	int NL = 11;
	int COMMENT = 12;
	int ID = 13;
	int TYPE = 14;
	int LPAREN = 15;
	int RPAREN = 16;
	int LBRACE = 17;
	int RBRACE = 18;
	int COLON = 19;
	int COMMA = 20;
	int DOT = 21;
	int MULT = 22;
	int PLUS = 23;
	int MINUS = 24;
	int DIV = 25;
	int MOD = 26;
	int SHARP = 27;
	int FLAT = 28;
	int ASGN = 29;
	int GE = 30;
	int LE = 31;
	int GT = 32;
```

```
int LT = 33;
int EQ = 34;
int NEQ = 35;
int AND = 36;
int OR = 37;
int NOT = 38;
int NUM = 39;
int NOTE = 40;
int STRING = 41;
int INSTRUMENT = 42;
int INSTRUMENT_NAME = 43;
int STATEMENT = 44;
int FORLOOP_STMT = 45;
int PRINT_STMT = 46;
int BLOCK_STMT = 47;
int FUNCT_CALL_EXPR = 48;
int FOR_EXPR = 49;
int PARAM_EXPR = 50;
int DEC_PARAM_EXPR = 51;
int FUNCT_STMT = 52;
int ID_EXPR = 53;
int MATH_TERM = 54;
int LEFT_EXPR = 55;
int RIGHT_EXPR = 56;
int NUMID_EXPR = 57;
int UMINUS = 58;
int DEC_EXPR = 59;
int DEC_ASGN = 60;
int LEFT_ASGN = 61;
int PLUS_MATH_TERM = 62;
int MINUS_MATH_TERM = 63;
int LITERAL_for = 64;
int LITERAL_foreach = 65;
int LITERAL_if = 66;
int LITERAL_else = 67;
int LITERAL_break = 68;
```

```
	int LITERAL_return = 69;
	int LITERAL_add = 70;
	int LITERAL_print = 71;
	int LITERAL_procedure = 72;
	int LITERAL_function = 73;
	int LITERAL_save = 74;
	int LITERAL_playSong = 75;
	int LITERAL_pitch = 76;
	int LITERAL_length = 77;
	int LITERAL_volume = 78;
}
```