MPL

Music Processing Language

LRM

Project Manager: Bo Wang (bw2450)

Language Guru: Yilin Xiong (yx2274)

System Architect: Shengyi Lin (sl3759)

System Integrator: Ying Tan (yt2443)

Verification and Validation: Mengting Wu (mw2987)

1. Lexical Elements

1.1 Token:

The tokens of MLP will be mainly C-like. There are five classes of tokens: identifiers, keywords, constants, operators and separators. Blank, tabs, newlines and comments will be ignored, except as they separate two consecutive tokens.

1.2 Comment:

The characters "/*" introduce a comment, which is terminated by the characters "*/". No nested comments is supported. The characters "//" introduce a comment with the scope of one line.The characters "/*", "*/", and "//" should follow a "\" to be included as a string or character literals.

1.3 Identifier:

An identifier is a sequence of characters for naming variables, functions and new data types. The sequence must start with a letter or the underscore character '_' in identifiers; all following characters can be any combination of letters, numbers, or the underscore character. Letters are the ASCII characters a-z and A-Z. Numbers are the composition of ASCII characters 0-9. And Identifiers are case sensitive.

1.4 Keywords:

The following identifiers are reserved as keywords, which may not be used otherwise:

double	**int**	**char**	**boolean**	**string**
Note	**Melody**	**Music**	**Track**	**break**
continue	**if**	**else**	**true**	**false**
while	**for**	**void**	**return**	

1.5 Constant:

Four kinds of constants are included with the corresponding data type:

Integer-constant:

An integer constant consists of a sequence of digits and is mainly C-like.

Double-constant:

A double constant consists of an integer part, a decimal part and a fraction part and is mainly C-like.

Character-constant:

A character constant is a sequence of one or more characters enclosed in single quotes and is mainly C-like.

String-constant:

A string constant, also called as a string literal, is a sequence of characters surrounded by double quotes and is mainly C-like.

Pitch-constant:

A pitch constant, is the constant value in integer for certain pitch. These constant could represent as [A-G](#|b)?[0-9] in regular expression, which is the scientific pitch notation. For example C3, Bb4.

Instrument-constant:

A instrument constant, is the constant value in integer for certain instrument. For example VIOLIN, PIANO.

1.6 Separator:

A separator separates tokens and is single-character tokens:

```
( ) [ ] { } \t \n , ;
```

1.7 Operator:

An operator is a special token that performs an operation on either one or two operands:

```
+ - * || && != == != < > <= >=
```

Full coverage of operators can be found in Expressions.

1.8 Types:

Identifiers, or names, refer to a variety of things: functions, tags of structures and variables and so forth. And different types, including both the basic types and the derived types, can define the variables.

1.8.1 Basic type:

There are several fundamental types in MPL: int, character, string, boolean, double, void, note, melody, track, and music.

Integer (int):

Integer variables have the natural size suggested by the host machine architecture. In MPL, all integers represent unsigned values unless specified otherwise.

Character (char):

Character variables store a single character among any member of the execution character set.

String (string):

String variables are large enough to store any sequence of combinations from the character set.

Boolean (Boolean):

Boolean variables hold the value of either “true” or “false”.

Double (double):

Double variables are the double precision floating point variables holding real numbers of specific precision.

Void (void):

Void type specifies an empty set of values. It is used as the type returned by functions that generate no value.

Note (Note):

Note variables are the object that specifies the unit of a piece of music with the states of pitch (int), starttime (int), duration (int), and strength (int).

Melody (Melody):

Melody variables are defined as a sequential list of notes, which can be declared by the keyword Melody.

Track (Track):

Track variables are containers of melodies with a specified attribute timbre (string).

Music (Music) :

Music variable is a sequential collection of tracks, which can be assigned with a string of name (string).

1.8.2 Derived type:

Besides the basic types, there is a conceptually infinite class of derived types constructed from the fundamental types in the following ways:

- *Arrays* of objects of a given type;
- *Functions* returning objects of a given type.

In general, these methods of constructing objects can be applied recursively.

2. Syntax Analysis:

2.1 Expression:

An expression consists of at least one operand and zero or more operators. Operands are typed objects such as constants, variables, and function calls that return values. An operator specifies an operation to be performed on its operand(s). Operators may have one, two, or three operands, depending on the operator.

2.1.1 Assignment Operators

=:

The standard assignment operator = simply stores the value of its right operand in the variable specified by its left operand. As with all assignment operators, the left operand (commonly referred to as the "lvalue") cannot be a literal or constant value.

Examples:

```
int a = 1;
```

```
string str = "Hello World!";
```

```
Note note1(C4);
Note note2 = note1;
```

Compound assignment operators perform an operation involving both the left and right operands, and then assign the resulting expression to the left operand. Here is a list of the compound assignment operators, and a brief description of what they do:

+=:

Add the left operand to the right operand, and then assign the result to the left operand, i.e. a += b means a = a + b.

Examples:

```
int a = 3;
a += 2;  // a is 5 now
```

```
Note note(C4);
note += 2;
// raise the pitch of Note by 2 and the pitch is D4 now
```

```
// Both of melody1 and melody2 are of type Melody
melody1 += melody2;
// concatenate melody1 and melody2 to a new Melody
```

```
// Both of music1 and music2 are of type Music
music1 += music2;
// concatenate music1 and music2 to a new Music
```

-=:

Subtract the right operand from the left operand, and then assign the result of the subtraction to the left operand. i.e. a -= b means a = a-b.

Examples:

```
int a = 2;
a -= 1;  // a is 1 now
```

```
Note note(D4);
note -= 2;  // lower the pitch of Note by 2 and the pitch is C4 now
```

***=:**

Multiply the left operand to the right operand, and then assign the result of the multiplication to the left operand. i.e. a *= b means a = a * b.

Examples:

```
int a = 2;
```

```
a *= 5;  // a is 10 now
```

```
// melody is of type Melody
// and it contains: 1 3 5 |
melody *= 3;  // melody now becomes:
              // 1 3 5 | 1 3 5 | 1 3 5
```

```
// Both melody1 and melody2 are of type Melody
// melody1 contains: 1 3
// melody2 contains: 3 5
melody1 *= melody2;  // melody1 now becomes:
                     // 1 3 |
                     // 3 6 |
```

2.1.2 Incrementing and Decrementing

++:

The increment operator ++ adds 1 to its operand. The operand must be an int. You can apply the increment operator either before or after the operand. This is used in the for_statement to increment the index.

--:

The increment operator -- subscribe 1 from its operand. The operand must be an int. You can apply the decrement operator either before or after the operand. This is used in the for_statement to decrement the index.

2.1.3 Arithmetic Operators

Table 1 shows the detail description for the definition of operations for each data types.

Table 1. Operations for each data types

int/double +/- int/double	Standard arithmetic add and subtract for real numbers. `a = 3 + 1;  b -= 1.0;`

Note +/- int	Raise/lower the pitch of Note by the value in Int. `noteNew = noteOld + 1;  note += 3;`
Note + Note	Combine these Notes, and become to Melody `mld = note1 + note2;`
Melody + Melody	Concatenate these Melodies into a new Melody `mldNew = mld1 + mld2;`
Music + Music	Concatenate these Music into a new Music `mscNew = msc1 + msc2;`
int/double * int/double	Standard arithmetic multiple for real numbers. `a = 3.3 * 2;  b *= 5;`
Melody * Int	Repeat the Melody for Int times to a new Melody `mldNew = mld * 4;  mld *= 4;`
Melody * Melody	Superimpose Melodies to a new Melody mldNew = mld1 * mld2
Music * Music	Superimpose all Tracks in these Music into a new Music mscNew = msc1 * msc2

2.1.4 Comparison Operators

Comparison operators are used to determine how two operands relate to each other. The result of the comparison operators is either 1 or 0, meaning true or false respectively. The comparison operators used in MPL are explained as follows:

==:

The equal-to operator == tests its two operands for equality. The result is 1 if the operands are equal, and 0 if the operands are not equal.

>:

The greater-than operator > tests if the left operand is larger than the right operand. The result is 1 if the left operand is larger than the right operand, and is 0 if not.

<:

The less-than operator < tests if the left operand is smaller than the right operand. The result is 1 if the left operand is smaller than the right operand, and is 0 if not.

!=:

The not-equal-to operator != tests its two operands for inequality. The result is 1 if the operands are not equal, and 0 if the operands *are* equal.

>=:

The greater-than-or-equal-to >= tests if the left operand is larger than or equal to the right operand. The result is 1 if the left operand is larger than or equal to the right operand, and is 0 if not.

<=:

The less-than-or-equal-to >= tests if the left operand is smaller than or equal to the right operand. The result is 1 if the left operand is smaller than or equal to the right operand, and is 0 if not.

2.1.5 Logical Operators

Logical operators test the truth value of a pair of operands.

&&:

The logical conjunction operator && tests if two expressions are both true. If the first expression is false, then the second expression is not evaluated.

||:

The logical conjunction operator || tests if at least one of two expressions it true. If the first expression is true, then the second expression is not evaluated.

!:

You can prepend a logical expression with a negation operator ! to flip the truth value

2.1.6 Array Subscripts

You can access array elements by specifying the name of the array, and the array subscript (or index, or element number) enclosed in brackets. Here is an example, supposing an array of Notes called Note:

Example:

```
Note notes[]  = {Note(C4), Note(D4), Note(E4)};
```

We can access the array element Note(E4) through notes[2].

2.1.7 Function Calls as Expressions

A call to any function which returns a value is an expression, which can be treated as a variable with the value of the returned value and with the type of the return value type.

Example:

```
int myFunction() {
    return 1;
}
int main(){
    int a = 1 + myFunction(); // a is 2
}
```

2.1.8 Operator Precedence

The precedence of operators can be shown below:

Precedence	*Operator Description*
1	Function calls, array subscripting
2	Logical negation and unary negation
3	Multiplication, addition and subtraction expressions
4	Greater-than, less-than, greater-than-or-equal-to, and less-than-or-equal-to expressions
5	Equal-to and not-equal-to expressions
6	Logical AND expressions
7	Logical OR expressions
8	All assignment expressions

2.2 Statement

2.2.1 if

You can use the if statement to conditionally execute part of your program, based on the true value of a given expression. Here is the general form of an if statement:

```
if(expression)
    statement
else
    statement
```

If test evaluates to true, then then-statement is executed and else-statement is not. If test evaluates to false, then else-statement is executed and then-statement is not. The else clause is optional.

Example:

```
if(note.getPitch() < C4) {
   note += 1;
} else {
   note -= 1;
}

// raise the pitch by 1 when the pitch is less
// than C4, else lower the pitch by 1
```

2.2.2 while

The while statement is a loop statement with an exit test at the beginning of the loop. Here is the general form of the while statement:

```
while(expression)
    statement
```

The while statement first evaluates test. If test evaluates to true, statement is executed, and then test is evaluated again. statement continues to execute repeatedly as long as test is true after each execution of statement.

Example:

```
int i = 0;
while(i < msc.numOfTracks()) {
    msc.track[i].setTimbre(VIOLIN);
```

```
    i++;
}
// set all tracks of the music to VIOLIN
```

2.2.3 for

The for statement is a loop statement whose structure allows easy variable initialization, expression testing, and variable modification. It is very convenient for making counter-controlled loops. Here is the general form of the for statement:

```
for(initialize; expression; step)
    statement
```

Example:

```
for(int i = 0; i < msc.numOfTracks(); i++) {
msc.track[i].setTimbre(VIOLIN);
}
// set all tracks of the music to VIOLIN
```

The for statement first evaluates the expression initialize i=0. Then it evaluates the expression test i < msc.numOfTracks(). If test is false which means all tracks in the music has been traversed, then the loop ends and program control resumes after statement. Otherwise, if test is true, then statement is executed to set the timbre of this track as VIOLIN. Finally, step is evaluated, and the next iteration of the loop begins with evaluating test again.

2.2.4 break

You can use the break statement to terminate a while, for, or switch statement.

Example:

```
int i = 0;
while(true) {
   msc.track[i].setTimbre(VIOLIN);
   i++;
   if(i >= msc.numOfTracks()) {
```

```
        break;
    }
}
// set all tracks of the musc to VIOLIN
```

2.2.5 continue

You can use the continue statement in loops to terminate an iteration of the loop and begin the next iteration.

Example:

```
for(int i = 0; i < msc.numOfTracks(); i++) {
    if(msc.track[i]) {
        continue;
    }
msc.track[i].setTimbre(VIOLIN);
}
// set tracks whose timbre isn't VIOLIN to VIOLIN
```

2.2.6 return

You can use the return statement to end the execution of a function and return program control to the function that called it.

Example:

```
int myFunction() {
    return 1;
}
// a function which returns 1
```

2.3 Function:

2.3.1 General function:

1. void setNoteDefault(int pitch = C4, int starttime = 0, int duration = 500, int strength = 100)

This method is used for setting the default values for the attributes of Note objects.

Example:

```
setNoteDefault(D1, 1000, 1000, 200);
```

2. int changeToMillisecond(int minute, int second = 0, int millisecond = 0)

This method is used for converting the time into millisecond.

Example:

```
int time = changeToMillisecond(2, 30, 50);
Note note = Note(C1, time, 200, 100);
```

3. void printTime(int millisecond)

This method is used for printing the time in minute/second/millisecond manner to stdout.

Example:

```
int time = changeToMillisecond(2, 30, 50);
printTime(time);  // The output should be: 2/30/50
```

4. Music read(string path)

This method is used for reading a midi file from specified path.

Example:

```
Music music = read("Path/inputFile.midi");
```

5. void write(Music music, string outputpath)

This method is used for exporting a Music object to a midi file with specified output path.

Example:

```
write(music, "Path/outputFile.midi");
```

6. void print(String str)

This method is used for printing string to stdout.

Example:

```
print("Hello World!");
```

7. int sizeof(Note notes[])

This function is used for getting the size of the Note array.

Example:

```
Note notes[] = {Note(C4), Note(D4)};
int s = sizeof(notes);
```

2.3.2 Note

The methods for Note class are shown below:

1. Note::Note(int pitch)

This constructor is used for constructing a new Note object with the specified pitch and other attributes set to default values.

Example:

```
Note note(C4);
```

2. Note::Note(int pitch, int duration)

This constructor is used for constructing a new Note object with the specified pitch and duration and other attributes set to default values.

Example:

```
Note note = Note(C4, 100);
```

3. Note::Note(int pitch, int duration, int starttime)

This constructor is used for constructing a new Note object with the specified pitch, duration and start time and strength set to default value.

Example:

```
Note note = Note(C4, 100, 200);
```

4. Note::Note(int pitch, int duration, int starttime, int strength)

This constructor is used for constructing a new Note object with the specified pitch, duration, start time and strength.

Example:

```
Note note = Note(C4, 100, 200, 100);
```

5. Note::Note(Note note)

This constructor is used for cloning an existing Note object to a new Note object.

Example:

```
Note note(C4, 100, 200, 100);
Note clone_note(note);
```

6. void Note::setDuration(int duration)

This method is used for setting the duration attribute of a Note object.

Example:

```
Note note(C4, 100, 200, 100);
note.setDuration(400);
```

7. int Note::getDuration()

This method is used for getting the duration attribute of a Note object.

Example:

```
Note note(C4, 100, 200, 100);
int duration = note.getDuration();
```

8. void Note::setPitch(int pitch)

This method is used for setting the pitch attribute of a Note object.

Example:

```
Note note(C4, 100, 200, 100);
note.setPitch(D4);
```

9. int Note::getPitch()

This method is used for getting the pitch attribute of a Note object.

Example:

```
Note note(C4, 100, 200, 100);
int pitch = note.getPitch();
```

10. void Note::setStartTime(int startTime)

This method is used for setting the startTime attribute of a Note object.

Example:

```
Note note(C4, 100, 200, 100);
note.setStartTime(200);
```

11. int Note::getStartTime()

This method is used for getting the startTime attribute of a Note object.

Example:

```
Note note(C4, 100, 200, 100);
int startTime = note.getStartTime();
```

12. void Note::setStrength(int strength)

This method is used for setting the strength attribute of a Note object.

Example:

```
Note note(C4, 100, 200, 100);
note.setStrength(200);
```

13. int Note::getStrength()

This method is used for getting the strength attribute of a Note object.

Example:

```
Note note(C4, 100, 200, 100);
int strength = note.getStrength();
```

2.3.3 Melody:

The methods for Melody class are shown below:

1. Melody::Melody()

This constructor is used for constructing a new Melody object.

Example:

```
Melody melody = Melody();
```

2. Melody::Melody(Note notes[])

This constructor is used for constructing a new Melody object with a Note array.

Example:

```
Note notes[] = {Note(C4), Note(D4)};
Melody melody = Melody(notes);
```

3. Melody::Melody(Melody melody)

This constructor is used for cloning an existing Melody object to a new Melody object.

Example:

```
Note notes[] = {Note(C4), Note(D4)};
Melody melody(notes);
Melody clone_melody = Melody(melody);
```

4. Melody Melody::subMelody(int startPos, int endPos)

This method is used for obtaining a new Melody object that is a piece of this Melody, beginning from the specified startPos and extending to the specified endPos inclusively.

Example:

```
Note notes[] = {Note(C4), Note(D4),
                Note(E4), Note(F4)};
Melody melody(notes);
// Melody: 1 2 3 |
// is returned
Melody sub_melody = melody.subMelody(0,2);
```

5. Note Melody::addNote(Note note)

This method is used for adding the specified Note object to the Melody object.

Example:

```
setNoteDefault(C4, 200, 0, 200);
Note notes[] = {Note(C4), Note(D4),
                Note(E4), Note(F4)};
Melody melody(notes);

setNoteDefault(C4, 200, 0, 200);
Note new_note(C4);

// The melody becomes 1 2 3 4|
//                    1      |
melody.addNote(new_note);
```

6. Note Melody::deleteNote(int index)

This method is used for deleting the specific Note object at the specified position in the Melody object.

Example:

```
Note notes[] = {Note(C4), Note(D4),
                Note(E4), Note(F4)};
Melody melody(notes);

// The melody becomes 1 2 0 4|
melody.deleteNote(2);
```

7. Note Melody::getNote(int index)

This method is used for getting the specific Note object at the specified position in the array of Note which composes the Melody object.

Example:

```
Note notes[] = {Note(C4), Note(D4),
                Note(E4), Note(F4)};
Melody melody(notes);

// Note(E4)is returned
Note note = melody.getNote(2);
```

8. Note[] Melody::getNotes()

This method is used for getting the Note array which composes the Melody.

Example:

```
Note notes[] = {Note(C4), Note(D4),
                Note(E4), Note(F4)};
Melody melody = Melody(notes);

// {Note(C4), Note(D4), Note(E4), Note(F4)}
// is returned
Note res_notes[] = melody.getNotes();
```

9. int Melody::getLength()

This method is used for getting the length of notes array which composes the Melody object.

Example:

```
Note notes[] = {Note(C4), Note(D4),
                Note(E4), Note(F4)};
Melody melody(notes);

int len = melody.getLength(); // 4 is returned
```

10. int Melody::getTimeLength()

This method is used for getting the total time length of the Melody object.

Example:

```
setNoteDefault(C4, 200, 0, 200);
Note notes[] = {Note(C4), Note(D4),
                Note(E4), Note(F4)};
Melody melody(notes);

// 800 is returned
int len = melody.getTimeLength();
```

2.3.4 Track:

The methods for Track class are shown below:

1. Track::Track()

This constructor is used for constructing a new Track object.

Example:

```
// an empty track is returned
Track track = Track();
```

2. Track::Track(Melody melody)

This constructor is used for constructing a new Track object based on the input Melody object.

Example:

```
Note notes[] = {Note(C4), Note(D4)};
Melody melody(notes);
Track track = Track(melody);
```

3. Track::Track(Melody melody, int timbre)

This constructor is used for constructing a new Track object based on the input Melody object and the timbre.

Example:

```
Note notes[] = {Note(C4), Note(D4)};
Melody melody(notes);

Track track = Track(melody, PIANO);
```

4. Track::Track(Track track)

This constructor is used for cloning an existing Track object to a new Track object.

Example:

```
Note notes[] = {Note(C4), Note(D4)};
Melody melody(notes);
Track track1(melody, PIANO);
Track track2 = Track(track1);
```

5. void Track::setTimbre(int timbre)

This method is used for setting the timbre attribute of a Track object.

Example:

```
Note notes[] = {Note(C4), Note(D4)};
Melody melody(notes);
Track track(melody, PIANO);

track.setTimbre(VIOLIN);
```

6. void Track::insertMelody(int startTime, Melody melody)

This method is used for insertting the Melody object at the specified startTime in Track.

Example:

```
setNoteDefault(C4, 200, 0, 200);
Note notes1[] = {Note(C4), Note(D4)};
Melody melody1(notes1);
Track track(melody1, PIANO);

setNoteDefault(C4, 200, 0, 200);
Note notes2[] = {Note(E4), Note(F4)};
Melody melody2(notes2);

track.insertMelody(400,melody2);
```

7. Melody Track::getMelody()

This method is used for getting the Melody object which composes the Track.

Example:

```
Note notes[] = {Note(C4), Note(D4)};
Melody melody(notes);
Track track = Track(melody);

Melody res_melody = track.getMelody();
```

8. int Track::getLength()

This method is used for getting the length of melody array which composes the Track object.

Example:

```
Note notes[] = {Note(C4), Note(D4)};
Melody melody(notes);
Track track = Track(melody);

int len = track.getLength(); // 2 is returned
```

2.3.5 Music

The methods for Music object are shown below:

1. Music::Music()

This constructor is used for constructing an new Music object.

Example:

```
Music music = Music();
```

2. Music::Music (Track [] tracks)

This constructor is used for constructing an new Music object based on the input Track array.

Example:

```
Note notes[] = {Note(C4), Note(D4)};
Melody melody(notes);
Track tracks[] = {Track(melody, PIANO), Track(melody, VIOLIN)};

Music music = Music(tracks);
```

3. Music::Music (Music music)

This constructor is used for cloning an existing Music object to a new Music object.

Example:

```
Note notes[] = {Note(C4), Note(D4)};
Melody melody(notes);
Track tracks[] = {Track(melody, PIANO)};
Music music(tracks);
Music clone_music = Music(music);
```

4. Track Music::getTrack(int index)

This method is used for getting the specific Track object at the specified position in the array of Track which composes the Music object.

Example:

```
Note notes[] = {Note(C4), Note(D4)};
Melody melody(notes);
Track tracks[] = {Track(melody, PIANO), Track(melody, VIOLIN)};
Music music(tracks);

Track res_track = music.getTrack(1);
```

5. Track[] Music::getTracks()

This method is used for getting the Track array which composes the Music.

Example:

```
Note notes[] = {Note(C4), Note(D4)};
Melody melody(notes);
Track tracks[] = {Track(melody, PIANO), Track(melody, VIOLIN)};
Music music(tracks);

Track res_tracks[] = music.getTracks();
```

6. int Music::getNumberOfTracks()

By using the getNumberOfTracks method, one can get the length of track array which composes the Music object.

Example:

```
Note notes[] = {Note(C4), Note(D4)};
Melody melody(notes);
Track tracks[] = {Track(melody, PIANO), Track(melody, VIOLIN)};
Music music(tracks);

int len = music.getNumberOfTracks();  // 2 is returned
```

7. int Music::getTimeLength()

By using the getTimeLength method, one can get the total time length of the Music object.

Example:

```
setNoteDefault(C4, 200, 0, 200);
Note notes[] = {Note(C4), Note(D4)};
Melody melody(notes);
Track track(melody, PIANO);
Music music(tracks);

// 400 is returned
int len = music.getTimeLength();
```

3. Full Grammar

The full grammar for MLP is shown below:

```
primary_expression
      : IDENTIFIER
      | constant
      | '(' expression ')'
      | TRUE
      | FALSE
      ;

constant
      : Integer-constant:
      | Double-constant:
      | Character-constant
      | String-constant
      | Pitch-constant
      | Instrument-constant
      ;

postfix_expression
      : primary_expression
      | postfix_expression '[' expression ']'
      | postfix_expression '(' ')'
      | postfix_expression '(' argument_expression_list ')'
      | postfix_expression '.' IDENTIFIER
      | postfix_expression ‘++’
      | postfix_expression ‘--’
      ;

argument_expression_list
      : assignment_expression
      | argument_expression_list ',' assignment_expression
      ;
```

```
unary_expression
      : postfix_expression
      | ‘++’ unary_expression
      | ‘--’ unary_expression
      | ‘-’ unary_expression
      | ‘!’ unary_expression
      ;

multiplicative_expression
      : unary_expression
      | multiplicative_expression '*' unary_expression
      ;

additive_expression
      : multiplicative_expression
      | additive_expression '+' multiplicative_expression
      | additive_expression '-' multiplicative_expression
      ;

comparison_expression
      : additive_expression
      | comparison_expression '<' additive_expression
      | comparison_expression '>' additive_expression
      | comparison_expression ‘<=‘ additive_expression
      | comparison_expression ‘>=‘ additive_expression

equality_expression
      : comparison_expression
      | equality_expression ‘==‘ comparison_expression
      | equality_expression ‘!=‘ comparison_expression
      ;

logical_and_expression
      : equality_expression
      | logical_and_expression ‘&&’ equality_expression
      ;
```

```
logical_or_expression
      : logical_and_expression
      | logical_or_expression '||' equality_expression
      ;

assignment_expression
      : logical_or_expression
      | unary_expression assignment_operator assignment_expression
      ;

assignment_operator
      : '='
      | '+='
      | '-='
      | '*='
      ;

declaration
      : type_specifier init_declaration_list ';'
      ;

init_declarator_list
      : init_declarator
      | init_declarator_list ',' init_declarator
      ;

init_declarator
      : declarator '=' initializer
      | declarator
      ;

initializer
      : '{' initializer_list '}'
      | '{' initializer_list ',' '}'
      | assignment_expression
      ;
```

```
initializer_list
      : initializer
      | initializer_list ',' initializer
      ;

type_name
      : int
      | double
      | boolean
      | string
      | char
      | Note
      | Melody
      | Track
      | Music
      | void
      ;

declarator
      : IDENTIFIER
      | '(' declarator')'
      | declarator '[' ']'
      | declarator '[' assignment_expression ']'
      | declarator '(' parameter_type_')'
      | declarator '(' ')'
      | declarator '(' identifier_list ')'
      ;

parameter_list
      : parameter_declaration
      | parameter_list ',' parameter_declaration
      ;

parameter_declaration
      : type_specifier declarator
      | type_specifier
```

```
        ;

identifier_list
        : 'IDENTIFIER
        | identifier_list ',' IDENTIFIER
        ;

statement
        : compound_statement
        | expresion_statement
        | selection_statement
        | iteration_statement
        | jump_statement
        ;

compound_statement
        : '{' '}'
        | '{' block_item_list '}'
        ;

block_item_list
        : block_item
        | block_item_list block_item
        ;

block_item
        : declaration
        | statement
        ;

expression_statement
        : ';'
        | expression ';'
        ;

selection-statement
```

```
      : IF '(' expression ')' statement
      | IF '(' expression ')' statement ELSE statement
      ;

iteration_statement
      : WHILE '(' expression ')' statement
      | FOR '(' expression_statement expression_statement ')'
      statement
      | FOR '(' expression_statement expression_statement expression
      ')' statement
      | FOR '(' declaration  expression_statement ')' statement
      | FOR '(' declaration  expression_statement  expression ')'
 statement

jump_statement
      : CONTINUE ';'
      | BREAK ';'
      | RETURN ';'
      | RETURN expression ';'
      ;
translation_unit
      : external_declaration
      | translation_unit external_declaration
      ;

external_declaration
      : function_definition
      | declaration
      ;

function_definition
      : type_specifier declarator declaration_list
      compound_statement
      | type_specifier declarator compound_statement
      ;
```

```
declaration_list
      : declaration
      | declaration_list declaration
      ;
```