MPL

Music Processing Language

Final Project

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)

Content

Chapter 1 White Paper

1.1 Introduction

Getting bored of recording your inspiration note by note on the paper? Want to know how does your music sound for some specified instrument directly? Instead of creating monotonous rhythm, want to create more attractive music? MPL is aimed at providing a non-traditional way for professional musicians or even beginners to create their own music easily and conveniently. Compositions can be completed with dynamics, arpeggios, accelerando, staccatos, repeats, and much more by simply using multiple libraries of existing data structures, operations and functions provided or create your own programmatically. Compose, record and arrange your song programmatically, your creativity can simply go wild by using a language like this!

1.2 Target Users

MPL is intended for both music amateurs and professional musicians, who have a passion for both music and programming. It allows you to create and process music in terms of notes, melodies, tracks or the entire piece of music in a convenient and interesting way.

If you are music amateurs or fans with merely basic music knowledge, we will provide you with libraries of existing data structures, operations and functions to compose your own music or conduct any level of recreation, including making adaptations of music with its pitch, timbre, chords, melody, tracks or the synthesis of different pieces of music. With MPL, you are free to immerse yourself into the world of music without the concerns of being an expert in both music and programming, and enjoy yourself to the ultimate.

For the music experts, on the other hand, you will be involved as both the users and the re-creators of MPL. With professional knowledge of music and a passion for programming, you can create your own functions with MPL to develop customized

musical tools for composition. By such means, your creative ideas can also facilitate and reach users of MPL who share the common interest in music.

1.3 Sparking Features

MPL helps users to create notes, tracks, melodies as well as music. What's more, users can do arithmetic on data structures to synthesize and make adaptation of music. Compared to many music languages, MPL simplifies the data structure to make it easy to use. So, users can do music edition, composition more conveniently in a programming approach.

MPL is designed to be extendible, which allows composers and music software developers to build their own functions by programming. So, they can apply their professional musical knowledge in it, and create their own musical tools, compositions and instruments.

MPL also includes some advanced functions, which can intelligently identify the chords of a melody which users provide, and help users to improve the complexity of composition so they can do professional music arrangement for a simple melody without needing to know anything about music. Such as, recommendation of the harmonies chords to the simple melody, swapping the instruments, varies the choruses. As long as you know the basic melody you want to convey, our language can algorithmically create a customized, unique piece of music, synced to your melody, all in under a minute.

Chapter 2 Language Tutorial

2.1 Getting Started

First, we will show how to write a program to print the words: hello, world to show the process of creating the program text, compiling it, loading it, running it and finding out where the output went. With these mechanical details mastered, everything else is comparatively easy. In MPL, the program to print “hello world” is

Example:

```
void main(String arg[]) {
     print(“Hello World!\n”);
}
```

To compile and run MPL language, it is require to have binary file -- “MPL” to translate and JAR file -- “MPL.jar” for environment. For simplification, “MPL.sh” implements all the process from translating, compiling to running. If you are under operating system like Linus, MacOS, or UNIX, you can compile and run your program with the command

```
/path/to/MPL/MPL.sh /path/to/MPL/ your_program.mpl result_name
```

If you are under an operating system not satisfying the requirement above, like Windows, you can run translating, compiling, and running separately:

```
/path/to/MPL/MPL your_program.mpl result_name
javac -cp /path/to/MPL/MPL.jar result_name.java
java -cp .:/path/to/MPL/MPL.jar result_name
```

If you haven't botched anything, such as omitting a character or misspelling something, the hello.mpl program will proceed silently, and make an executable file called hello. If you run hello by typing the command

```
./MPL.sh ./ hello.mpl hello
```

It will print : **Hello World!**

Now, for some explanations about the program itself, a MPL program, whatever its size, consists of functions and variables. A function contains statements that specify the computing operations to be done, and variables store values used during the computation. Our functions are mainly C like, and our example is a function named main. Normally you are at liberty to give functions whatever names you like, but "main" is special - your program begins executing at the beginning of main. This means that every program must have a main somewhere and it will usually call other functions to help perform its job.

A function is called by naming it, followed by a parenthesized list of arguments, so this calls the function print with the argument "Hello World!\n". The print() is a function that prints output, in this case the string of characters between the double quotes.

A sequence of characters in double quotes, such as "Hello world!\n", is a string constant or string literal in MPL. And the sequence \n in the string is the C-like notation for the newline character, when printed advances the output to the left margin is on next line.

2.2 Note Declarations and Initialization

The basic component of music is its notes. To create a new note, we can use the declaration statement as follows:

Example:

```
Note note1 = Note(); // Create a new note named note1
```

The basic type Note has four attributes pitch, duration, starttime, strength. To set the default value of the new created note, we can use the statement as follows :

Example:

```
setNoteDefault(C4, 250, 0, 200);
// set pitch default value as C4, set duration value as
// 250, set startime value as 0, set strength value as 200
```

After setting the default values manually, once we create a new note again by the declaration

Example:

```
Note note2 = Note(); // Create a new note named note2
```

Then the attribute values of note2 will be the manually-set default values instead of the systematically-set default values.

To initialize Note, we can use the initialization statements as follows:

Example:

```
Note(G4);
// create a note with the pitch value as G4, and the values
// of its duration, starttime, strength as default

Note(G4,500);
```

```
// create a note with the pitch value as G4, the duration
// value as 500, and the values of its starttime,strength
// as default

Note(G4,500,10,300);
// create a note with the pitch value as G4, the duration
// value as 500, the starttime value as 10, and the
// strength value as 300
```

2.3 Melody Declarations and Initialization

Melody in MPL is defined as a sequential list of notes, which can be manipulated as a whole. To create a new melody, we can use the declaration statement as follows.

Example:

```
Melody melody0 = Melody() // construct a new melody object
```

To initialize a melody, we can use the initialization statements as follows:

Example:

```
Melody(notes)
// construct a new Melody object  composed of the input the
// note array notes[]

Melody(melody0)
// construct a new Melody object which has same attributes
// as the input melody0.
```

2.4 Melody Modification

2.4.1 Melody Addition

User can use the operation: + to overlay the new music on the original one. In this case, we assume that the following forms of original music and new music:

Original music:

```
Piano:  1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - ||
Violin: 1 - 5 - | 6 - 5 - | 4 - 3 - | 2 - 1 - ||
```

New music:

```
Piano:  1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - | 5 5 4 4 |
        3 3 2 - | 5 5 4 4 | 3 3 2 - ||
Violin: 1 - 5 - | 6 - 5 - | 4 - 3 - | 2 - 1 - | 5 - 4 - |
        3 - 2 - | 5 - 4 - | 3 - 2 - ||
```

The example to combine these music are shown below:

```
void main(){

    String path = "twinkle_twinkle.mid";
    Music music = read(path);
    // read in the midi file from path

    Track tracks[] = music.getTracks(); //get tracks array

    /* 5 5 4 4 | 3 3 2 - | 5 5 4 4 | 3 3 2 - ||*/

    setNoteDefault(G4, 250, 0, 200);
```

```
    // set pitch default value as G4, set duration
    // value as 250, set startime value as 0, set
    // strength value as 200

    Note notes0[] = {Note(), Note(), Note(F4),  Note(F4),
        Note(E4),  Note(E4), Note(D4, 500), Note(), Note(),
        Note(F4),  Note(F4),  Note(E4),  Note(E4),
        Note(D4, 500)};
   // initialize notes array
   /* 5 - 4 - | 3 - 2 - | 5 - 4 - | 3 - 2 - ||*/
   setNoteDefault(G4, 250, 0, 200);
   // set pitch default value as G4, set duration value
   // as 250, set startime value as 0, set strength value
   // as 200

   Note notes1[] = {Note(), Note(F4), Note(E4),  Note(D4),
       Note(), Note(F4), Note(E4), Note(D4)};
   // initialize notes array

   Melody melody0 = Melody(notes0);// Initialize melody0
   Melody melody1 = Melody(notes1);//Initialize melody1

   tracks[0].getMelody() += melody0;
   // concatenate melody0 to tracks[0]'s original melody
   tracks[1].getMelody() += melody1;
   // concatenate melody1 to tracks[1]'s original melody

   write(music, "new_twinkle_twinkle.mid");
   // write the new music to output path
}
```

2.4.2 Melody Multiplication

User can use the operation * to superimpose melodies to a new melody. In this case, we assume that the following forms of original music and new music:

Original music:

```
Piano:  1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - | 5 5 4 4 |
Violin: 1 - 5 - | 6 - 5 - | 4 - 3 - | 2 - 1 - | 5 - 4 - |

Piano:  3 3 2 - | 5 5 4 4 | 3 3 2 - ||
Violin: 3 - 2 - | 5 - 4 - | 3 -  2 -||
```

New music:

```
Piano:  1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - | 5 5 4 4 |
Violin: 1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - | 5 5 4 4 |
        1 - 5 - | 6 - 5 - | 4 - 3 - | 2 - 1 - | 5 - 4 - |

Piano:  3 3 2 - | 5 5 4 4 | 3 3 2 - ||
Violin: 3 3 2 - | 5 5 4 4 | 3 3 2 - ||
        3 - 2 - | 5 - 4 - | 3 - 2 - ||
```

The example to create the new music is shown below:

```
void main(){
   string path = "twinkle_twinkle.mid";
   Music music = read(path); //read in the music midi file

   Track tracks[] = music.getTracks();
   // tracks[0] is for the melody with timbre value
   // of Piano and tracks[1] is of the melody with
   // timbre value of Violin
```

```
   // Original melody:
   // 1 - 5 - | 6 - 5 - | 4 - 3 - | 2 - 1 - |
   // 5 - 4 - | 3 - 2 - | 5 - 4 - | 3 - 2 - ||

   // After multiplication:
   // 1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - |
   // 1 - 5 - | 6 - 5 - | 4 - 3 - | 2 - 1 - |
   //
   // 5 5 4 4 | 3 3 2 - | 5 5 4 4 | 3 3 2 - ||
   // 5 - 4 - | 3 - 2 - | 5 - 4 - | 3 -  2 -||

   tracks[1].getMelody() *= tracks[0].getMelody();
   // get the melody of the piano track and superimpose
   // it to the melody of the violin track to set as
   // the new melody of the violin track

   write(music, "new_twinkle_twinkle.mid");
   // export to a midi file
}
```

2.4.3 Melody Modification

2.4.3.1 Note Insertion

User can use the function addNote() to insert single note into the existing melody with user-specified attributes. In this case, the note to be inserted is defined as follows:

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

The original music is set by the following statement as the following form:

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

Original Melody:

```
1 2 3 4||
```

The example of inserting a note into the original melody is shown below:

```
melody.addNote(new_note);
```

After insertion, the new melody will be as follows:

New Melody :

```
// The melody becomes 1 2 3 4|
//                    1      |
```

2.4.3.2 Note Deletion

This method is used for deleting the specific Note object at the specified position in the Melody object. The index should be smaller than the note array's length, and larger than 0 inclusively:

The original melody is set by the following statement as the following form:

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

So we get the following melody: `1 2 3 4||`

The example to delete a specified note from the melody is shown below:

Example:

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

2.5 Track Declaration and Initialization

Track in MPL is defined as the container of melodies with a specified attribute named timbre. To create a new track ,we can use the declaration statement as follows:

Example:

```
Track track1 = Track(); //construct a new track object
```

To initialize a track, we can use the initialization statements as follows:

Example:

```
Track(melody0);
// construct a new Track object based on melody0 with the
// default timbre as PIANO

Track(melody0, VIOLIN);
// construct a new Track object based on melody0 and set the
// timbre as VIOLIN

Track(track1);
// construct a new Track object which has same attributes as track1
```

2.6 Music Declaration and Initialization, Reading and Writing

To create a new music, we can use the declaration statement as follows:

Example:

```
Music music = Music(); // construct a new Music object
```

To initialize a piece of music, we can use the initialization statements as follows:

Example:

```
Music(tracks);
// construct a new Music object based on the tracks[] array

Music(music0);
// clone an existing Music object to a new Music object
```

MPL supports importing external midi files directly for further modification, the function read() can be applied to read files with the statements as follows:

Example:

```
string path = "twinkle_twinkle.mid";
Music music = read(path); // read in the midi file from path
```

After finishing the music composition, you can export your music with the function write() as a midi file to play in the midi player. The statement is as follows:

Example:

```
write(music, "new_twinkle_twinkle.mid");
// write the new music to output path
```

2.7 Music Creation

The example of writing a complete music program is shown below: In this program, we want to write a piece of music such as:

```
Piano:  1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - ||
Violin: 1 - 5 - | 6 - 5 - | 4 - 3 - | 2 - 1 - ||
```

The program below shows how to write it.

```
void main(){

    // The following two-line statements are used for
    // initialize required objects for the music:
    /* 1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - || */

    setNoteDefault(C4, 250, 0, 200);
    // set pitch default value as C4, set duration value as 250,
    // set startime value as 0, set strength value as 200


    Note notes0[] = {Note((), Note(), Note(G4), Note(G4),
         Note(A4), Note(A4),Note(G4, 500), Note(F4), Note(F4)
         Note(E4), Note(E4),Note(D4), Note(D4), Note(C4, 500)};
    // Initialize notes array



     // The following two-line statements are used for initialize
     // required objects for the music:
```

```
    /* 1 - 5 - | 6 - 5 - | 4 - 3 - | 2 - 1 - || */

    setNoteDefault(C4, 500, 0, 200);
    // set pitch default value as C4, set duration value as
    // 250, set starttime value as 0, set strength value as 200


    Note notes1[] = {Note(), Note(G4), Note(A4),Note(G4),
                  Note(F4), Note(E4),Note(D4), Note()};
    // Initialize notes array for the violin part

    Melody melody0 = Melody(notes0), melody1 = Melody(notes1);
    // Initialize melody object

    Track tracks[] = {Track(melody0, PIANO),
    Track(melody1, VIOLIN)};
    // Initialize track[] array, which combines the track
    // from melody0 object with PIANO timbre, and another
    // track from melody1 object with VIOLIN timbre.

    Music music = Music(tracks); // Initialize music object

    write(music, "twinkle_twinkle.mid");
    // output a midi file.
}
```

We provide the ways for user to create music conveniently. Not only user can write a piece of music, but can also modify music in different approaches, such as: music addition, music multiplication and change timbre of track.

2.7.1 Music Addition

User can use the arithmetic operator + to concatenate several pieces of music to a new music. In this case, we assume that the following forms of original music and new music:

Original music1:

```
Piano: 1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - ||
```

Original music2:

```
Piano:  5 5 4 4 | 3 3 2 - | 5 5 4 4 | 3 3 2 - ||
Violin: 5 5 4 4 | 3 3 2 - | 5 5 4 4 | 3 3 2 - ||
```

New music:

```
Piano:  1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - | 0 - - - |
        0 - - - | 0 - - - | 0 - - - ||
Piano:  0 - - - | 0 - - - | 0 - - - | 0 - - - | 5 5 4 4 |
        3 3 2 - | 5 5 4 4 | 3 3 2 - ||
Violin: 0 - - - | 0 - - - | 0 - - - | 0 - - -| 5 - 4 - |
        3 - 2 - | 5 - 4 - | 3 -  2 -||
```

The example to create the new music is shown below:

```
void main(){
    string path0 = "twinkle_twinkle0.mid";
    string path1 = "twinkle_twinkle1.mid";
    Music music0 = read(path1); // reading music0
    Music music1 = read(path2); // reading music1

    Music music = music0 + music1;
    // concatenate music0 and music1 to create a new music.
```

```
    // music0 is played first then follows music1

    write(music, "new_twinkle_twinkle.mid");
    // export the midi file
}
```

2.7.2 Music Multiplication

User can use the operation * to superimpose musics to a new music. In this case, we assume that the following forms of original music and new music:

Original music1:

```
Piano: 1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - ||
```

Original music2:

```
Violin: 1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - ||
Piano:  1 - 5 - | 6 - 5 - | 4 - 3 - | 2 - 1 - ||
```

New music:

```
Piano:  1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - ||
Violin: 1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - ||
Piano:  1 - 5 - | 6 - 5 - | 4 - 3 - | 2 - 1 - ||
```

The example to create the new music is shown below:

```
void main(){
    string path0 = "twinkle_twinkle0.mid";
    string path1 = "twinkle_twinkle1.mid";
    Music music0 = read(path1); // reading music0
```

```
    Music music1 = read(path2); // reading music1
    Music music = music0 * music1;
    // superimpose music0 and music1 together to create a new music

    write(music, "new_twinkle_twinkle.mid");
    // export the midi file
}
```

2.7.3 Change Timbre

User can use the method setTimbre() to change the timbre of the entire melody to create a new music. In this case, we assume that the following forms of original music and new music:

Original music:

```
Piano: 1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - ||
Piano: 1 - 5 - | 6 - 5 - | 4 - 3 - | 2 - 1 - ||
```

Original music:

```
Violin: 1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - ||
Violin: 1 -  5 -  | 6 - 5 -  | 4 - 3 -  | 2 - 1 -  ||
```

The example to create the new music is shown below:

```
void main(){
    Music music = read("twinkle_twinkle.mid");
    //reading music

    for(int i = 0; i < music.getNumberOfTracks(); i++) {
```

```
        music.getTrack(i).setTimbre(VIOLIN);
    }
   // for each track in the music, set its timbre to VIOLIN

   write(music, "new_twinkle_twinkle.mid");
   // export the midi file
}
```

Chapter 3 Language Reference Manual

3.1 Lexical Elements

3.1.1 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.

3.1.2 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.

3.1.3 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.

3.1.4 Keywords:

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

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

3.1.5 Constant:

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

- ***Integer-constant****:* An integer constant consists of a sequence of digits.

- ***Double-constant****:* A double constant consists of an integer part, a decimal part and a fraction part 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:*** An instrument constant, is the constant value in integer for certain instrument, such as VIOLIN, PIANO.

3.1.6 Separator:

A separator separates tokens and is single-character tokens:

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

3.1.7 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.

3.1.8 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.

3.1.8.1 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.

- ***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).

3.1.8.2 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.

3.2. Syntax Analysis:

3.2.1 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.

3.2.1.1 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 = Note(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 = 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 = 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 |
```

3.2.1.2 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.

3.2.1.3 Arithmetic Operators

Table 1 shows the detail description for the definition of 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 + Note	Concatenate a Melody and a Note into a new Melody mldNew = mld + note;
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

Table 3-1. Operations for each data types

3.2.1.4 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 true if the operands are equal, and false if the operands are not equal.

>:

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

<:

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

!=:

The not-equal-to operator != tests its two operands for inequality. The result is true if the operands are not equal, and false 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 true if the left operand is larger than or equal to the right operand, and is false 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 true if the left operand is smaller than or equal to the right operand, and is false if not.

3.2.1.5 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

3.2.1.6 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].

3.2.1.7 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
}
```

3.2.1.8 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

3.2.2 Statement

3.2.2.1 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
```

3.2.2.2 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.getNumberOfTracks()) {
    Track currentTrack = msc.getTrack(i);
    currentTrack.setTimbre(VIOLIN);
     i++;
}
// set all tracks of the music to VIOLIN
```

3.2.2.3 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.

3.2.2.4 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
```

3.2.2.5 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
```

3.2.2.6 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
```

3.2.3 Function:

3.2.3.1 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);
```

3.2.3.2 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();
```

3.2.3.3 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();
```

3.2.3.4 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
```

3.2.3.5 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 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();
```

3.3. Full Grammar

The full grammar for MLP is shown below:

```
constant:
   INT_CON
   | DOUB_CON
   | BOOL_CON
   | STR_CON
   | INS_CON
   | PIT_CON

primary_expr:
   ID
   | constant
   | LPA expr RPA
   | primary_expr LPA arg_expr_list RPA
   | primary_expr LPA  RPA
   | primary_expr LBRK expr RBRK
   | primary_expr DOT ID
   | primary_expr ADD2
   | primary_expr SUB2

unary_expr:
   primary_expr
   | ADD2 unary_expr
   | SUB2 unary_expr
   | SUB unary_expr
   | NOT unary_expr

mul_expr:
```

```
   unary_expr
   | mul_expr MUL unary_expr

add_expr:
   mul_expr
   | add_expr ADD mul_expr
   | add_expr SUB mul_expr

cmpr_expr:
   add_expr
   | cmpr_expr LT add_expr
   | cmpr_expr GT add_expr
   | cmpr_expr LEQ add_expr
   | cmpr_expr GEQ add_expr

eql_expr:
   cmpr_expr
   | eql_expr EQ cmpr_expr
   | eql_expr NEQ cmpr_expr

and_expr:
   eql_expr
   | and_expr AND eql_expr

or_expr:
   and_expr
   | or_expr OR and_expr

ass_op:
   ASS
   | AASS
```

```
   | SASS
   | MASS

expr:
   or_expr
   | unary_expr ass_op expr

arg_expr_list:
   expr
   | arg_expr_list COM expr

statement:
   SEMI
   | expr SEMI
   | LBRE RBRE
   | LBRE statement_list RBRE
   | DATATYPE declarator_list SEMI
   | IF LPA expr RPA statement
   | IF LPA expr RPA statement ELSE statement
   | WHILE LPA expr RPA statement
   | FOR LPA statement statement expr RPA statement
   | FOR LPA statement statement RPA statement
   | CONTINUE SEMI
   | BREAK SEMI
   | RETURN SEMI
   | RETURN expr SEMI

statement_list:
   statement
   | statement_list statement
```

```
val_declarator:
   ID
   | val_declarator LBRK or_expr RBRK
   | val_declarator LBRK RBRK

init:
   expr
   | DATATYPE LPA RPA
   | DATATYPE LPA arg_expr_list RPA
   | LBRE initializer_list RBRE

initializer_list:
   init
   | initializer_list COM init

declarator_list:
   val_declarator
   | val_declarator ASS init
   | declarator_list COM val_declarator
   | declarator_list COM val_declarator ASS init

param:
   DATATYPE val_declarator

para_list:
   param
   | para_list COM param

func_declarator:
   ID LPA para_list RPA
   | ID LPA RPA
```

```
function_definition:
   DATATYPE func_declarator LBRE statement_list RBRE

program:
   program statement
   | program function_definition
```

Chapter 4 Project Plan (written by Bo Wang)

4.1 Project Processes

4.1.1 Planning

Project planning is an important first step for the whole project. Without careful time management, specific task responsibility arrangement and efficient communication, it will be hard to cooperate smoothly and efficiently among team members. Thus, the first thing our team did is to use the gantt chart to make a rough time management at the very beginning with the aid of SmartChart. Also we made sure to meet consistently each week on Friday to discuss common problems and arrange the new task for next week. Secondly, according to the skills and past experience of our members, we allocated different tasks and responsibilities to our team members. Thirdly, to ensure efficient communication, we used Google docs and Github to allow members of the group to work independently and then merged work.

4.1.2 Specification

We followed the ANSI C specification from Dennis Ritchie's C reference manual. We chose to use Ocaml for the MPL part. And the target code is Java code.

4.1.3 Development

We used Github as a distributed version control system to allow all members of the group to work independently. At the early stage of the development, two people are responsible for Ocaml, two people are responsible for Java and one person participates in both two parts to ensure the further work involving these two parts efficiently, such as the java code generation part. We used an iterative approach to software development and each group member worked on an individual feature at a time, merging the branch when all regression tests passed. During the development, we developed on several

branches to implement features such as the type checking, java code generation and merged them to the whole branch when it was tested ensuring the correctness and reliability.

4.1.4 Testing

The testing process of Ocaml part and Java part are done separately at the early stage of the development. On the high level, at the end of each stage of development, every group member wrote unit test to ensure her part of the cord worked correctly and reliably. At the second stage, in both Ocaml part and Java part some closely related codes are merged to test to ensure performance. At the third stage, both Ocaml part and Java part were finished. At the Ocaml side, several test case written in MPL were tested to generate Java Code correctly. At the Java side, several test case to implement functions of our language were tested written in Java to ensure the correct performance. At the fourth stage, these two parts are merged together to test the whole process.

4.2 Roles and Responsibilities

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)

4.3 Styling Guide

The styling rules for our program are as follows:

(1) Maximum length of a single line must not exceed 80 characters

(2) One statement per line

(3) Break up complex function into smaller functions as much as possible

(4) Use easy-understanding name for functions

4.4 Timeline

The Gantt Chart is made using smartsheet that can be viewed through the following link: https://app.smartsheet.com/b/publish?EQBCT=65b07f7a00a64cd3a855c315ed61e47a

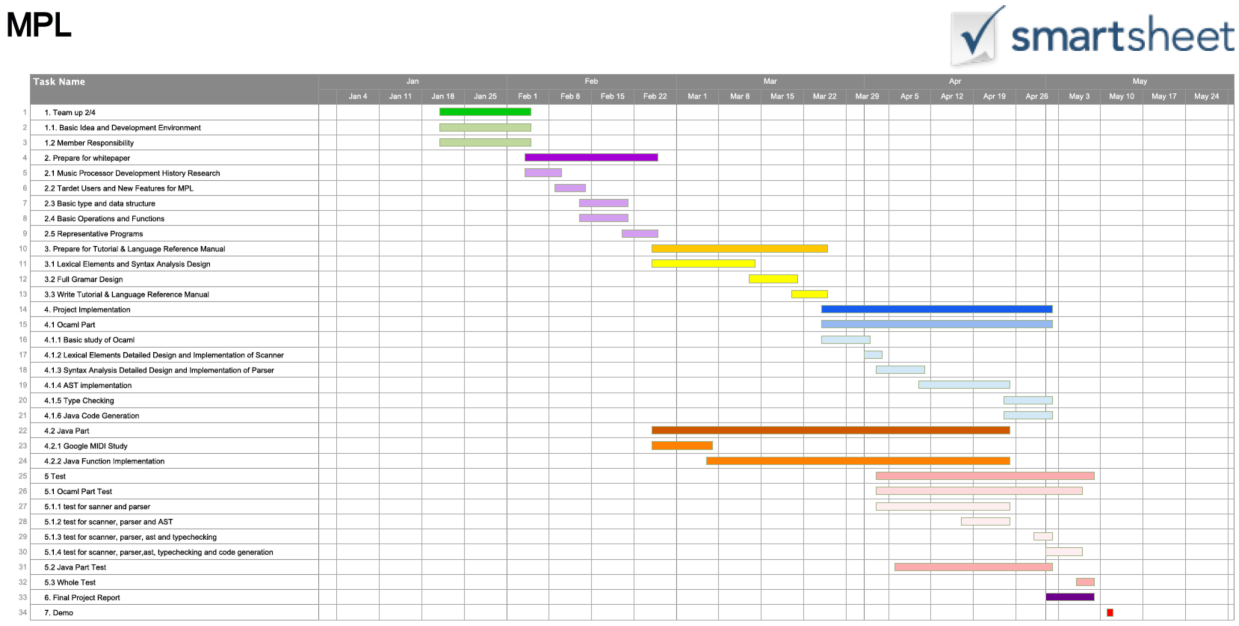
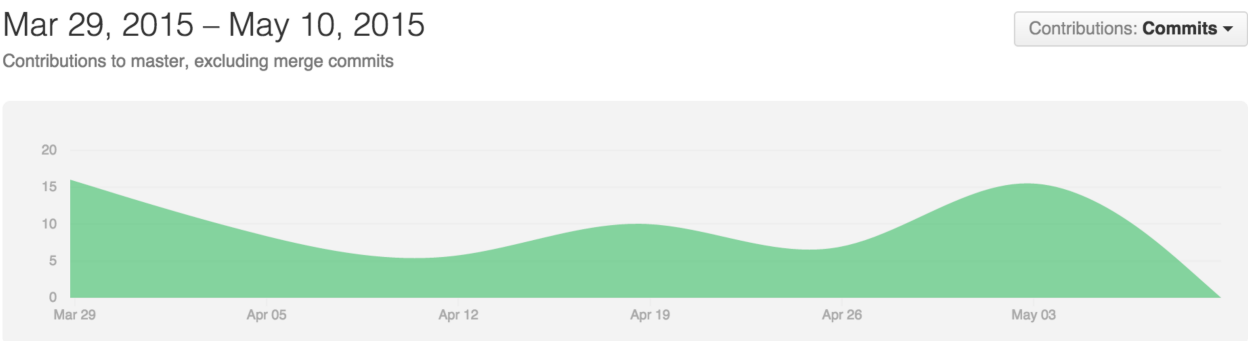


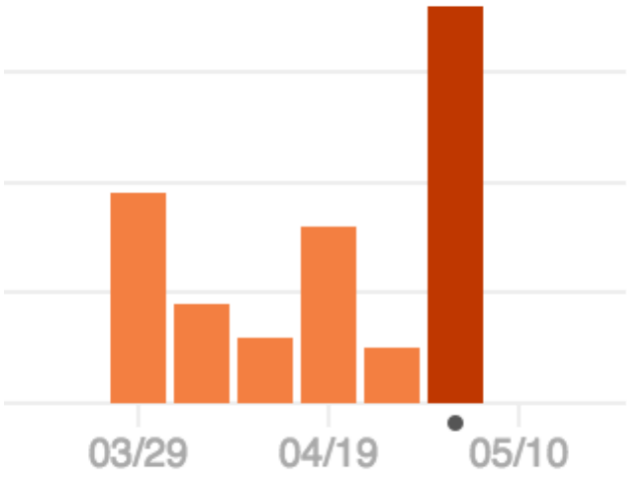
Figure 4-1 Gantt Chart showing our work timeline

4.5 Project Log

The following graphs are retrieved from Github history, showing our truly working intensity during the project. Details are on https://github.com/PLT-MPL/MPL.git



(a) Commit time based on day.



(b) Commit time based on week.

Figure 4-2 The commit times to in during the project.

Chapter 5 Language Evolution (written by Mengting Wu)

5.1 Evolution

Initially, our language was developed for implementing only the “hello world” program, which only incorporates the simple version for parsing the function definition and function call based on basic data types. And then we incrementally delivered new features for the compiler to parse for more complicated programs until the full grammar was entirely covered. And we tried to maintain the attributes consistent by following the process of lexical, syntax and semantic analysis, and integrating our language with Java code based on the bound type and values for each pair of the constructors in our language.

5.2 Compiler Tools

As for the compiler tools, we used the OCam language to implement translation from our language to Java code. It is targeted for developing applications that involve symbolic computation for compilers and interpreters. The lexer, parser, ast, checker and javagen are implemented in OCaml for lexical analysis, syntax analysis, semantic analysis, type checking and Java code generation respectively.

5.3 Compiling Libraries

In the compiler constructing, we used the OCaml standard library and compiled it with the OCaml standard tools.

5.4 Compiling Libraries

We divided our compiler development into lexical, syntax, semantic analysis and code generation, and implemented different phases of compiling for our language respectively according to our LRM. The lexical features are implemented in consistency with the

specifications of the tokens we use in our language. The syntax features in the LRM are covered with the implementation and moderate modification for our full grammar. And their equivalent semantics are achieved with the help of the construction of the abstract syntax tree together with the specifications of set of semantic rules.

Chapter 6 Translator Architecture (written by Bo Wang)

6.1 Overview

MPL compiler transforms s MPL program into a playable MIDI file. The whole process can be described as the following flow chart.

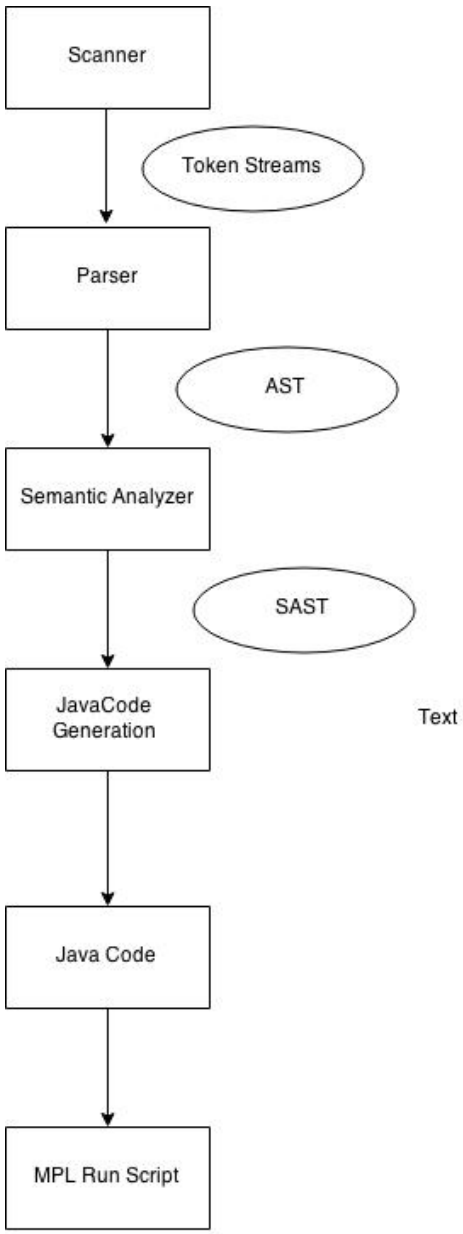


Figure 6-1 The flow chart for the project.

The compiler scans the source file, parses the resulting tokens, creates an abstract syntax tree, checks the abstract syntax tree semantically. Translates the tree into intermediate code and finally translates this intermediate representation into a MIDI file which can be played in most media players.

6.2 Interfaces between Modules

6.2.1 Scanner

The scanner tokenizes the input into readable units that will be used by the parser. During this process involves discarding whitespace and comments. The scanner was written with ocamllex.

6.2.2 Parser

The parser generates an abstract syntax tree from the tokens provided by the scanner to depict the grammatical structure of the token streams. Syntax errors are caught here. The scanner was written with ocamlyacc.

6.2.3 Semantic Analyzer

Semantic Analyzer takes in the abstract syntax tree given by the parser and performs semantic checks, outputting a semantic abstract syntax tree. Type checking is did in this process.

6.2.4 Code Generator

This module walks the SAST to generate the target Java Code mapping functions and operations defined in MPL to the corresponding functions and operations in java code that will implement these functions.

6.3 Task Assignment

(1) Scanner (Bo Wang, Mengting Wu)

(2) Parser (Yilin Xiong)

(3) AST (Bo Wang, Mengting Wu)

(4) Typechecking (Yilin Xiong, Ying Tan)

(5) Javacode Generation (Yilin Xiong, Bo Wang, Mengting Wu)

(6) Javacode Implementation (Ying Tan, Shengyi Lin)

(7) Testing (Shengyi Lin)

Chapter 7 Development Environment and Runtime

(written by Ying Tan)

7.1 Development Environment

The MPL team used OCamllex and OCamlyacc to generate the lexical analyzer and the parser for MPL. In addition, we used OCaml 4.02.1 to do semantic analysis (including type checking) for MPL programs, and generate target codes into Java. To support the actual running of the target programs and achieve the effect of music composition, the MIDI Library (https://github.com/LeffelMania/android-midi-lib) was introduced into our project. We also used JDK 1.6 and Eclipse to develop a Java package which provides API callable in the target code of a MPL program. Lastly, Git hosted on GitHub were used to support collaborative development and version control, and bash scripts and Makefile were used to ease the work of compiling and testing. At the end of the compilation process, a Java source code program is generated for each MPL program. The Java program is then translated into a runnable by the standard Java compiler embedded in JDK.

7.2 Runtime Environment

MPL supports a variety of environments including mac OS X, Ubuntu and windows 7, 8. The runtime environment of MPL including Ocaml, and JDK 1.6(JVM), and a MPL.jar.

Users can download the latest Ocaml from https://ocaml.org/. For mac user, ocaml can be installed just using the brew command: brew install ocaml. More details can be found at https://ocaml.org/docs/install.html.

To run the MPL, users need JDK 6 or later. If you are using MacOS, Java is built-in. If you are using Linux, make sure to use either the Sun JDK or OpenJDK. If you are using Windows, just download and install the latest JDK package. After installing JDK, be sure to have the java and javac commands in the current path(you can check this by typing java -version and javac -version at the shell prompt). More details about JDK can be found at http://www.oracle.com/technetwork/java/index.html.

What's more, users need MPL.jar, which is a jar library that can be downloaded at https://github.com/PLT-MPL/MPL/tree/master/ShellScript to generate midi file. This jar is generated from the target code. Users should add MPL.jar to the classpath. Mac users can use the command likes export CLASSPATH=$CLASSPATH:/path/to/MPL.jar

Besides, we provide users with a shell script: MPL.sh, which helps user to test MPL code more conveniently. To use it, user has to put the shell script and MPL.jar under the same directory. The usage command of the script is shown below, where the input is the MPL file, and output is the java target code.

```
<Path/To/MPL.sh> <Path/of/MPL.sh> <Input/Filename>  <execname>
```

Chapter 8 Scripts and Test Plan (written by Shengyi Lin)

8.1 Prerequisites

An executable file, MPL, is generated from five ocaml files -- lexer, parser, ast, checker and javagen. This executable file takes the role of lexical analysis, semantics analysis, type checking and target code generation.

An MPL API, MPL.jar, is generated from our JAVA code. To compile and run the target JAVA code, MPL.jar should be added to classpath.

8.2 Scripts

A shell script, MPL.sh, is written to make testing easier and more convenient. It integrate the procedure of target JAVA code generation, target JAVA code compilation and execution.
The usage of the script is
MPL.sh <Path/To/ShellScript/Folder> <Input/MPL/Filename>
<Output/Executable/Filename>

8.3 Test Plan

8.3.1 Test Procedure

The language was tested module by module. A HelloWorld MPL program is tested first. Then, the initialization and the behaviors of each structure are tested respectively. Finally, self-defined function and loops are tested.

8.3.2 Test modules

- Hello World
- Note Initialization
- Note Insertion
- Note Deletion
- Melody Initialization
- Melody Addition
- Track Initialization
- Music Initialization
- Music Addition
- Music Creation
- Self-defined Function

8.3.3 Test Examples

8.3.3.1. Hello World:

MPL Code:

```
void main(string arg[]) {
   print ("Hello World!\n");
}
```

Java Code:

```
import java.io.*;
import java.util.*;
import definition.*;
```

```
import function.PublicFunction;

public class HelloWorld {
    public static void main(String arg[]){
        PublicFunction.print("Hello World!\n");
    }
}
```

8.3.3.2. Music Creation

MPL Code:

```
void main(string arg[]) {

   // The following two-line statements are used for
   // initialize required objects for the music:
   /* 1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - || */

   setNoteDefault(C4, 250, 0, 200);
   // set pitch default value as C4, set starttime value as 0,
   // set duration value as 250, set strength value as 200

   Note notes0[] = {Note(), Note(), Note(G4), Note(G4),
      Note(A4), Note(A4),Note(G4, 500), Note(F4), Note(F4),
      Note(E4), Note(E4),Note(D4), Note(D4), Note(C4, 500)};
   // Initialize notes array

   // The following two-line statements are used for initialize
   // required objects for the music:
```

```
   /* 1 - 5 - | 6 - 5 - | 4 - 3 - | 2 - 1 - || */

   setNoteDefault(C4, 500, 0, 200);
   // set pitch default value as C4, set duration value as 500,
   // set starttime value as 0, set strength value as 200

   Note notes1[] = {Note(), Note(G4), Note(A4),Note(G4),
   Note(F4), Note(E4),Note(D4), Note()};
   // Initialize notes array for the violin part

   Melody melody0 = Melody(notes0), melody1 = Melody(notes1);
   // Initialize melody object

   Track tracks[] = {Track(melody0, PIANO), Track(melody1, VIOLIN)};
   // Initialize track[] array, which combines the track
   // from melody0 object with PIANO timbre, and another
   // track from melody1 object with VIOLIN timbre.

   Music music = Music(tracks); // Initialize music object

   write(music, "twinkle_twinkle0.mid");
   // output a midi file.

   print("Done...\n");

}
```

Java Code:

```
import java.io.*;
import java.util.*;
```

```
import definition.*;
import function.PublicFunction;

public class MusicCreation
{
    public static void main(String arg[]){
        PublicFunction.setNoteDefault(C.C4, 250, 0, 200);

        List<Note> notes0 = new ArrayList<Note>(Arrays.asList(
            new Note(), new Note(), new Note(C.G4),
            new Note(C.G4), new Note(C.A4), new Note(C.A4),
            new Note(C.G4, 500), new Note(C.F4),
            new Note(C.F4), new Note(C.E4), new Note(C.E4),
            new Note(C.D4), new Note(C.D4),
            new Note(C.C4, 500)));
        PublicFunction.setNoteDefault(C.C4, 500, 0, 200);

        List<Note> notes1 = new ArrayList<Note>(Arrays.asList(
            new Note(), new Note(C.G4), new Note(C.A4),
            new Note(C.G4), new Note(C.F4), new Note(C.E4),
            new Note(C.D4), new Note()));
        Melody melody0 = new Melody(notes0),
            melody1 = new Melody(notes1);
        List<Track> tracks = new ArrayList<Track>(
            Arrays.asList(new Track(melody0, C.PIANO),
            new Track(melody1, C.VIOLIN)));

        Music music = new Music(tracks);
        PublicFunction.write(music, "twinkle_twinkle0.mid");
        PublicFunction.print("Done...\n");
```

```
    }
}
```

8.3.3.3. Self-Defined Function

MPL Code:

```
Melody addChords1(Melody melody){

  Melody newMelody = Melody();
  for(int i = 0; i < melody.getLength(); i++){
     Note note = melody.getNote(i);
     int pitch = note.getPitch() - 12;
     int duration = note.getDuration();
     int starttime = note.getStartTime();
     int strength = note.getStrength();
     Note chord = Note(pitch, duration, starttime, strength);
     newMelody.addNote(chord);
  }
  return newMelody;
}

Melody addChords2(Melody melody){

  Melody newMelody = Melody();
  int i = 0;
  while(true){
     Note note = melody.getNote(i);
     int pitch = note.getPitch() - 12;
     int duration = note.getDuration();
```

```
      int starttime = note.getStartTime();
      int strength = note.getStrength();
      Note chord = Note(pitch, duration, starttime, strength);
      newMelody.addNote(chord);
      i++;
      if(i >= melody.getLength()) break;
   }
   return newMelody;
}


void main(string arg[]) {

   Music music = read("twinkle_twinkle3.mid");

   //reading music
   Melody melody = music.getTrack(0).getMelody();

   Melody newMelody = addChords1(melody);
   Track tracks[] = {music.getTrack(0), Track(newMelody, PIANO)};
   Music newMusic = Music(tracks);

   // for each track in the music, set its timbre to VIOLIN
   write(newMusic, "new_twinkle_twinkle.mid");
   // export the midi file

   print("Done...\n");

}
```

Java Code:

```
import java.io.*;
import java.util.*;
import definition.*;
import function.PublicFunction;

public class SelfDefinedFunction
{
    public static Melody addChords1(Melody melody){
        Melody newMelody = new Melody();
        for(int i = 0;i < melody.getLength(); i++){
            Note note = melody.getNote(i);
            int pitch = note.getPitch() - 12;
            int duration = note.getDuration();
            int starttime = note.getStartTime();
            int strength = note.getStrength();
            Note chord = new Note(pitch, duration,
                starttime, strength);
            newMelody.addNote(chord);
    }
    return newMelody;
}

    public static Melody addChords2(Melody melody){
        Melody newMelody = new Melody();
        int i = 0;
        while(true){
            Note note = melody.getNote(i);
            int pitch = note.getPitch() - 12;
            int duration = note.getDuration();
```

```
            int starttime = note.getStartTime();
            int strength = note.getStrength();
            Note chord = new Note(pitch, duration,
                starttime, strength);
            newMelody.addNote(chord);

            i++;
            if (i >= melody.getLength())break;
        }
        return newMelody;
    }

    public static void main(String arg[]){
        Music music = PublicFunction.read(
            "twinkle_twinkle3.mid");
        Melody melody = music.getTrack(0).getMelody();
        Melody newMelody = addChords1(melody);
        List<Track> tracks = new ArrayList<Track>(
            Arrays.asList(music.getTrack(0),
            new Track(newMelody, C.PIANO)));
        Music newMusic = new Music(tracks);
        PublicFunction.write(newMusic,
            "new_twinkle_twinkle.mid");

        PublicFunction.print("Done...\n");
    }
}
```

Chapter 9 Conclusion

9.1 Lessons learnt as a team

This is the first time for all our team members to work in such a large team to work on such challenging and exciting work. From building up the team with not much knowledge among all other team members to finally finishing the whole project perfectly, we learnt so much during the whole process, not only the knowledge about building up our own language, but also how to cooperate with all team members to work efficiently.

From this project, we learnt how to implement the front end of the compiler using Ocaml and how to implement functions of our language by translating it into targeted Java code. The whole process gives us great opportunities to learn things by ourselves and to put the theory knowledge into practice.

Also, from this project, we learnt much experience on how to work in a project team. Team productivity depends on how well people responsible for the different modules of the project work cooperatively with each other. Arranging task and allocating responsibility wisely, making careful and flexible project time plan in advance and during the whole process, efficient communicate between each team member and making each member participating into the project are all important keys to make success.

9.2 Lessons learnt by team members

Bo Wang:

This is the first time for me to be a project manager for a 5-people work, which gives me a challenging and exciting experience. To be a team project member, the core task is to make a careful time plan for the whole project and make sure that the plan can be executed well. Thus, with the aid of the project management knowledge I learnt from my undergraduate course, I used Gantt Chart to make a rough time plan at the beginning of the project. However, during the process, we found that it was hard to execute the plan strictly because of many factors. For example, we haven't learnt that much yet when we wanted to implement some certain steps of the compiler. Under these situations, we

changed our plan a little bit to do some self-learning and some other parts instead. From this project, I learnt that project time plan is not what we should only do at the very beginning, making a careful and flexible time plan should be an important task during the whole project process.

Secondly, I learnt that arranging task and allocating responsibility wisely, efficient communicate between each team member and making each member participating into the project are important keys to make success. At the early stage, I assigned three members: Bo Wang, Yilin Long, Mengting Wu to be responsible for complier implement part (Ocaml Part) and assigned two members Shengyi Lin, Ying Tan to be responsible for Java implement part. Then, when some small modules were implemented and could be tested, I further assigned one person in each part to be a tester. These two testers worked cooperatively for the final test at the end stage.

Thirdly, project manager should be a person responsible for coordinating different parts of the project. This is especially important when we want to merge different modules together, such as the Ocaml part and Java part in our project. Project Manager needs to build a communication bridge between people responsible for two parts and has a clear clue about how to make these two parts together efficiently. Thus, though my main work is on Ocaml part building scanner, constructing AST, I communicated with the other two members responsible for Java part also. This makes our work on writing Java Code Generation much more efficient later.

Yilin Xiong:

To ensure the consistency, I have much communication with my teammates, to ensure they understanding on our design were the same. This experience really improves my skill on corporation.

This project must be one of the most interesting projects. The purpose and process of this project is one that I have never experienced. From this project, I got to know the basic theory of programming languages, and implement these theories into developing our own programming language.

In these projects my job is mainly about syntax and semantic analysis and ensure the language consistency with our design. When implement syntax and semantic analysis,

we choose Ocaml as the language tool. This is my fist time getting touch with functional language. By playing with it, I got deeper knowledge about functional language and lambda expression. Writing the parser for the language is really a fun staff. Since then, I started to notice the interesting language grammar definition in different languages. Type checking and code generator are also one challenge our logical thinking.

Ying Tan:

It is a great experience working with a team on a compiler project and to see how the compilers translates a source code to a target code through lexical analysis, parsing analysis, semantic analysis , type checking and code generation. Though Ocaml is really a difficult language to understand, I learned how to use this to implement these phrases more quickly and conveniently compared to implementing these by pure c++ code.

Open to changes. Most of my jobs are related to operating, creating midi's structure and providing these functions for code generation part. I need to make my code flexible, so when there are some changes like changes of operation definitions from front end, I can adjust java functions without too many changes. Moreover, since I am a big fun of music, it is really enjoyable to create MPL's data structures such as: Note, Track, Melody, Music etc, as well as different music operation functions for them. Although some advanced intelligent functions have not been implemented in time, we will still keep trying it to improve MPL in the future.

Coding together is really useful when work as a team. Our team always gets together to do this project on every Friday afternoon, and we keep communicating with teammates in this process, and everyone knows what going on of other parts. Besides, we define the interface between different modules very clearly and keep each module as loose as possible. Thus, as an integrator in MPL team, I do not spend too much time to integrate different parts together.

Test cases should be built earlier. I write the test cases in java part, which tests different operations of midi, however, I test these cases a little bit late.

Mengting Wu:

It's been a great and challenging experience learning a new language for developing the compiler. And writing and testing a program incrementally is of great importance. Fixing bugs for hundreds of lines of code as a whole could be quite painful, especially when debugging in OCaml where few error information was given in feedback and different files were blended in execution as different phases of compiling are tightly integrated together. So it is with the increment implementation of the complicated features of the language starting with a simple prototype.

Shengyi Lin:

I have learnt much about how to work efficiently in a big team and how to schedule the project to avoid huge tasks before deadlines. In the past, I usually did projects individually or sometimes did project in a group of two or three people. Normally, I just needed to finish the projects or my parts by myself and did not need to communicate much with others. This is the first time that I work in such a big team. We hold meetings once a week, and discuss the problems we have met together as well as the progress of our own work. Every time when I first saw our new tasks, I was feeling that there were so much thing to do and we must need to spend a lot of time on finishing the tasks. But actually, when we discussed the problems together, separated the tasks into five parts, and did some work every week, the tasks were not so much or tough. It is really a great experience to work in a team with all team members working hard.

I have also learnt about how to write parser in Ocaml language and been more familiar with the structure of the JAVA language. Although the parser is written by my teammates, I also read and try to understand all their codes. By doing this, I learnt Ocaml and have a better sense of the parser now. My first programming language is C++, and I was not used to Java previously. But after this project, I can now better understand Java and use it more efficiently.

9.3 Advice for future team

For the future team, we will suggest they'd better not make their language too complex when they designed it. According to the limited time and knowledge we had at the start,

designing things too complex may make it hard to start the project and along the process, you may be found overwhelmed. Starting things as simple as possible and then after gaining some experience, adding more features into your language may be a wise choice to do.

9.4 Advice for Instructor

At first, please let us thank you for Professor Alfred V.Aho's effort for providing us such detailed and well-organized courses during this semester. Also, thank you for his great patience and helpful advice for our project. The only advice we could think of is that adding more background knowledge of Computer Science Theory as a review during this course could be much more helpful for us to learn more deeply and thoroughly of Programming and Language Translator course.

Chapter 10 Code Listing

Lexer:

```
let Pitch_cons = (['A'-'G']('#' | 'b')?['0'-'9'])
let Integer_cons = ['0'-'9']+
let Double_cons = ['0'-'9']+['.']['0' - '9']+ | ['0'-'9']+['.'] | ['.']['0' - '9']+
let String_cons = '\"' ('\\'_ | [^'\"'])* '\"'
let Identifier = ['a'-'z' 'A'-'Z'] ['_' 'a'-'z' 'A'-'Z' '0'-'9'])*
let Instrument_cons = "PIANO"| "VIOLIN"


rule token = parse
[' ' '\t' '\r' '\n']{ token lexbuf }
| "/*"              { comment lexbuf }
| "//"               { comment_newline lexbuf }
| '('            { LPA }
| ')'            { RPA }
| '['            { LBRK }
| ']'            { RBRK }
| '{'            { LBRE }
| '}'            { RBRE }
| ';'            { SEMI }
| ','                { COM }
| '.'                { DOT }
| '='                { ASS }
| "+="               { AASS }
| "-="               { SASS }
| "*="               { MASS }
| '+'            { ADD }
| "++"               { ADD2 }
| '-'            { SUB }
| "--"               { SUB2 }
| '*'            { MUL }
| '/'            { DIV }
```

```
| '!'            { NOT }
| "||"           { OR }
| "&&"           { AND }
| "=="           { EQ }
| "!="           { NEQ }
| '<'            { LT }
| '>'            { GT }
| "<="           { LEQ }
| ">="           { GEQ }

| "int"          { DATATYPE("int") }
| "string"           { DATATYPE("String") }
| "boolean"      { DATATYPE("Boolean") }
| "double"       { DATATYPE("double") }
| "void"             { DATATYPE("void") }
| "Note"             { DATATYPE("Note") }
| "Melody"           { DATATYPE("Melody") }
| "Track"            { DATATYPE("Track") }
| "Music"            { DATATYPE("Music") }

| "return"           { RETURN }
| "if"               { IF }
| "else"             { ELSE }
| "for"              { FOR }
| "while"            { WHILE }
| "break"            { BREAK }
| "continue"         { CONTINUE }

| "true"|"false"     as lxm { BOOL_CON(bool_of_string lxm) }
| Integer_cons       as lxm { INT_CON(int_of_string lxm) }
| Double_cons        as lxm { DOUB_CON(float_of_string lxm) }
| Pitch_cons     as lxm { PIT_CON(lxm) }
| Instrument_cons    as lxm { INS_CON(lxm) }
| String_cons        as lxm { STR_CON(lxm) }
| Identifier     as lxm { ID(lxm) }
```

```
  | eof               { EOF }
  | _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

 and comment = parse
 "*/" {token lexbuf}
 | _  {comment lexbuf}

 and comment_newline = parse
 '\n' {token lexbuf}
 | _  {comment_newline lexbuf}
```

Parser:

```
{%
    open Ast
%}

%token SEMI LPA RPA LBRK RBRK LBRE RBRE EOF DOT
%token COM ASS AASS SASS MASS ADD SUB MUL DIV ADD2 SUB2
%token NOT OR AND EQ NEQ LT GT LEQ GEQ
%token IF ELSE WHILE FOR
%token RETURN BREAK CONTINUE

%token <string> STR_CON
%token <string> DATATYPE
%token <bool> BOOL_CON
%token <int> INT_CON
%token <float> DOUB_CON
%token <string> INS_CON
%token <string> PIT_CON
%token <string> ID

%start program
%type <Ast.program> program
```

```
%%

constant:
     | INT_CON                { IntCon $1 }
     | DOUB_CON               { DoubleCon $1 }
     | BOOL_CON               { BoolCon $1 }
     | STR_CON                { StrCon $1 }
     | INS_CON                { InsCon $1 }
     | PIT_CON                { PitCon $1 }


primary_expr:
     | ID                          { Id $1 }
     | constant                        { $1 }
     | LPA expr RPA                    { Paren_Expr $2 }
     | primary_expr LPA arg_expr_list RPA  { Func_Call($1 , Some($3)) }
     | primary_expr LPA  RPA       { Func_Call($1, None) }
     | primary_expr LBRK expr RBRK     { Array($1,$3)}
     | primary_expr DOT ID             { Dot_Expr($1,$3) }
     | primary_expr ADD2               { Self_Add ($1) }
     | primary_expr SUB2               { Self_Sub ($1)}

unary_expr:
     | primary_expr           { $1 }
     | ADD2 unary_expr        { Self_Add2 ($2) }
     | SUB2 unary_expr        { Self_Sub2 ($2) }
     | SUB unary_expr     { Uminus($2) }
     | NOT unary_expr     { Not_Unary($2) }


mul_expr:
     | unary_expr             { $1 }
     | mul_expr MUL unary_expr { Multi($1,$3) }
```

```
add_expr:
    | mul_expr                { $1 }
    | add_expr ADD mul_expr { Add_Bin($1,$3) }
    | add_expr SUB mul_expr { Sub($1,$3) }



cmpr_expr:
    | add_expr                    { $1 }
    | cmpr_expr LT add_expr       { Less($1,$3) }
    | cmpr_expr GT add_expr       { Greater($1,$3) }
    | cmpr_expr LEQ add_expr  { Less_Equal($1,$3) }
    | cmpr_expr GEQ add_expr  { Greater_Equal($1,$3) }

eql_expr:
    | cmpr_expr                   { $1 }
    | eql_expr EQ cmpr_expr       { Equal($1,$3) }
    | eql_expr NEQ cmpr_expr  { Non_Equal($1,$3) }

and_expr:
    | eql_expr                { $1 }
    | and_expr AND eql_expr   { And($1,$3) }

or_expr:
    | and_expr                { $1 }
    | or_expr OR and_expr{ Or($1,$3) }

ass_op:
    | ASS                     { ASS }
    | AASS                    { AASS }
    | SASS                    { SASS }
    | MASS                    { MASS }

expr:
    | or_expr                         { $1 }
    | unary_expr ass_op expr  { Expr($1,$2,$3) }
```

```
arg_expr_list:
    | expr                          { [$1] }
    | arg_expr_list COM expr  { $3 :: $1 }

statement:
    SEMI                 { Stmt (None) }
    | expr SEMI              { Stmt (Some($1))}
    | LBRE RBRE              { Bre_Stmt(None) }
    | LBRE statement_list RBRE        { Bre_Stmt(Some(List.rev $2)) }
    | DATATYPE declarator_list SEMI { Dec($1,$2)}
    | IF LPA expr RPA statement                    {If($3,$5)}
    | IF LPA expr RPA statement ELSE statement  {If_else($3,$5,$7)}
    | WHILE LPA expr RPA statement                 {While($3,$5) }
    | FOR LPA statement statement expr RPA statement {For_complete($3,$4,$5,$7)}
    | FOR LPA statement statement RPA statement  {For_part($3,$4,$6)}
    | CONTINUE SEMI              { CONTINUE }
    | BREAK SEMI                 { BREAK }
    | RETURN SEMI                { RETURN }
    | RETURN expr SEMI    { Return($2) }

statement_list:
    | statement                  { [$1] }
    | statement_list statement    { $2 :: $1 }

val_declarator:
    ID                               { VId $1 }
    | val_declarator LBRK or_expr RBRK    { Array_Dec($1,Some($3)) }
    | val_declarator LBRK RBRK    {  Array_Dec($1,None) }

init:
    | expr                       { Expr_Init $1 }
    | DATATYPE LPA RPA               { Init($1, None)}
    | DATATYPE LPA arg_expr_list RPA { Init($1, Some($3)) }
```

```
    | LBRE initializer_list RBRE  { Func_Init($2) }

initializer_list:
    init                              { [$1] }
    | initializer_list COM init   { $3 :: $1}

declarator_list:
    | val_declarator                          { Val_Decl $1 }
    | val_declarator ASS init             { Assignment($1, $3) }
    | declarator_list COM val_declarator  { Dec_list($1,$3) }
    | declarator_list COM val_declarator ASS init  { Assign_list($1,$3,$5) }

param:
    DATATYPE val_declarator               { Param($1,$2) }

para_list:
    | param                                   { [$1] }
    | para_list COM param { $3 :: $1 }

func_declarator:
    | ID LPA para_list RPA    { Func_dec_p($1,Some($3))}
    | ID LPA RPA { Func_dec_p($1,None ) }



function_definition:
    DATATYPE func_declarator LBRE statement_list RBRE  { Func_Def($1,$2,List.rev
$4 )}


program:
    { [], [] }
    | program statement {($2 :: fst $1), snd $1 }
    | program function_definition { fst $1, ($2 :: snd $1) }
}
```

AST:

```
type ass_op =
      | ASS
      | AASS
      | SASS
      | MASS


 type expr =
      | Id of string
      | IntCon of int
      | DoubleCon of float
      | BoolCon of bool
      | StrCon of string
      | InsCon of string
      | PitCon of string
      | Dot_Expr of expr * string
      | Self_Add of expr
      | Self_Sub of expr
      | Self_Add2 of expr
      | Self_Sub2 of expr
      | Uminus of expr
      | Not_Unary of expr
      | Multi of expr * expr
      | Add_Bin of expr * expr
      | Sub of expr * expr
      | Less of expr * expr
      | Greater of expr * expr
      | Less_Equal of expr * expr
      | Greater_Equal of expr * expr
      | Equal of expr * expr
      | Non_Equal of expr * expr
      | And of expr * expr
      | Or of expr * expr
      | Expr of expr * ass_op * expr
```

```
    | Paren_Expr of expr
    | Array of expr * expr
    | Func_Call of expr * expr list option

type val_declarator =
    | VId of string
    | Array_Dec of val_declarator * expr option

type init =
    | Expr_Init of expr
    | Init of string * expr list option
    | Func_Init of init list

type declarator_list =
    | Val_Decl of val_declarator
    | Assignment of val_declarator * init
    | Dec_list of declarator_list * val_declarator
    | Assign_list of declarator_list * val_declarator * init

type statement =
    | Stmt of expr option
    | Bre_Stmt of statement list option
    | Dec of string * declarator_list
    | If of expr * statement
    | If_else of expr * statement * statement
    | While of expr * statement
    | For_complete of statement * statement * expr * statement
    | For_part of statement * statement * statement
    | CONTINUE
    | BREAK
    | RETURN
    | Return of expr

type param =
    Param of string * val_declarator
```

```
type func_declarator =
    | Func_dec_p of string * param list option

type function_definition =
    Func_Def of string * func_declarator * statement list

type program = statement list * function_definition list

let rec string_of_ass_op  = function
    | ASS ->  "="
    | AASS -> "+="
    | SASS -> "-="
    | MASS -> "*="

let rec string_of_expr = function
    | Id(s) -> s
    | IntCon(c) -> string_of_int c
    | DoubleCon(d) -> string_of_float d
    | BoolCon(b) -> string_of_bool b
    | StrCon(s) -> s
    | Dot_Expr(e,s) -> string_of_expr e ^ "." ^ s
    | Self_Add(e) -> string_of_expr e ^ "++"
    | Self_Sub(e) -> string_of_expr e ^ "--"

    | Self_Add2(e) -> "++" ^ string_of_expr e
    | Self_Sub2(e) -> "--" ^ string_of_expr e
    | Uminus(e) -> "-" ^ string_of_expr e
    | Not_Unary(e) -> "!" ^ string_of_expr e

    | Multi(e1,e2) -> "(" ^ string_of_expr e1 ^ ")" ^ "*" ^ "(" ^ string_of_expr
e2 ^ ")"

    | Add_Bin(e1,e2) -> string_of_expr e1 ^ "+" ^ string_of_expr e2
    | Sub(e1,e2) -> string_of_expr e1 ^ "-" ^ string_of_expr e2
```

```
    | Less(e1,e2) -> string_of_expr e1 ^ "<" ^ string_of_expr e2
    | Greater(e1,e2) -> string_of_expr e1 ^ ">" ^ string_of_expr e2
    | Less_Equal(e1,e2) -> string_of_expr e1 ^ "<=" ^ string_of_expr e2
    | Greater_Equal(e1,e2) -> string_of_expr e1 ^ ">=" ^ string_of_expr e2

    | Equal(e1,e2) -> string_of_expr e1 ^ "==" ^ string_of_expr e2
    | Non_Equal(e1,e2) -> string_of_expr e1 ^ "!=" ^ string_of_expr e2

    | And(e1,e2) -> string_of_expr e1 ^ "&&" ^ string_of_expr e2

    | Or(e1,e2) -> string_of_expr e1 ^ "||" ^ string_of_expr e2

    | Expr(e1,e2,e3) -> string_of_expr e1 ^ " " ^ string_of_ass_op e2 ^ " " ^
string_of_expr e3
    | Paren_Expr(e1) -> string_of_expr e1
    | Array(e1,e2) -> string_of_expr e1 ^ "[" ^ string_of_expr e2 ^ "]"
    | Func_Call(e1,Some(a)) -> string_of_expr e1 ^ "(" ^ String.concat ",
"(List.map string_of_expr a) ^ ")"
    | Func_Call(e1,None) -> string_of_expr e1 ^ "(" ^ ")"

let rec string_of_val_declarator = function
    | VId(s) -> s
    | Array_Dec(d,Some(e)) -> string_of_val_declarator d ^ "[" ^ string_of_expr
e ^ "]"
    | Array_Dec(d,None) -> string_of_val_declarator d ^ "[" ^ "]"

let rec string_of_init = function
    | Expr_Init(e) -> string_of_expr e
    | Init(s,None)  -> s ^ "(" ^ ")"
    | Init(s, Some(e)) -> s ^ "(" ^ String.concat ", " (List.map string_of_expr
e) ^ ")"
    | Func_Init(e)  -> "{" ^ String.concat "," (List.map string_of_init e) ^ "}"
```

Checker:

```
open Ast
open Table




 exception Not_Found of string
 exception Type_Not_Match of string
 exception Need_Init of string
 exception No_Main of string

 let current_scope = ref 0
 let inc_scope (u:unit) =
     current_scope := !current_scope + 1;
     !current_scope

 let rec call_list func env lst =
     match lst with
         [] -> fst env
         | h :: t ->
             let new_table = func env h in
             let env = (new_table, snd env) in
                 call_list func env t

 let find_type_by_name env s =
     let table = fst env in
     let scopes = snd env in
     let i = ref 0 in
     let scope = ref (List.hd scopes) in
     if (Hashtbl.mem table s) then
         let typeFunc = Hashtbl.find table s in
         String.sub typeFunc 5 ((String.length typeFunc) - 5)
     else begin
```

```
        while not (Hashtbl.mem table ((string_of_int (!scope)) ^ "_" ^ s)) do
            i := !i + 1;
            if !i >= List.length scopes then
                raise (Not_Found ("Could not find varialble " ^ s ^ "."));
            scope := List.nth scopes !i
        done;
        Hashtbl.find table ((string_of_int !scope) ^ "_" ^ s)
    end

let rec check_expr env expr =
    match expr with
        Id(s) -> find_type_by_name env s
        | IntCon(c) -> "int"
        | DoubleCon(d) -> "double"
        | BoolCon(b) -> "Boolean"
        | StrCon(s) -> "String"
        | InsCon(s) | PitCon(s) -> "int"
        | Dot_Expr(e,s) ->
            let typeID = check_expr env e in
            let table = fst env in
            if ((not (typeID = "Note")) && (not (typeID = "Music")) && (not (typeID
= "Melody")) && (not (typeID = "Track"))) then
                raise (Type_Not_Match ("The type of \"" ^ string_of_expr e ^ "\"
was expected of type Note, Melody, Track, or Music."));
            if not (Hashtbl.mem table (typeID ^ "_" ^ s)) then
                raise (Not_Found ("Could not find the function \"" ^ string_of_expr
e ^ "." ^ s ^ "\"."));
            let typeFunc = Hashtbl.find table (typeID ^ "_" ^ s) in
            String.sub typeFunc (String.length typeID + 1) ((String.length
typeFunc) - 1 - String.length typeID)

        | Self_Add(e) | Self_Sub(e) | Self_Sub2(e) | Self_Add2(e) ->
            let typeID = check_expr env e in
            if not (typeID = "int") then
                raise (Type_Not_Match ("The type of \"" ^ string_of_expr e ^ "\"
```

```
was expected of type int."));
            typeID

        | Not_Unary(e) ->
            let typeID = check_expr env e in
            if not (typeID = "Boolean") then
                raise (Type_Not_Match ("The type of \"" ^ string_of_expr e ^ "\"
was expected of type boolean."));
            typeID

        | Uminus(e) ->
            let typeID = check_expr env e in
            if (not (typeID = "int")) && (not (typeID = "double")) then
                raise (Type_Not_Match ("The type of \"" ^ string_of_expr e ^ "\"
was expected of type int or double."));
            typeID

        | Multi(e1,e2) ->
            let typeE1 = check_expr env e1 in
            let typeE2 = check_expr env e2 in
            if (((typeE1 = "int") && (typeE2 = "int")) || ((typeE1 = "double") &&
(typeE2 = "double")) ||
                ((typeE1 = "Melody") && (typeE2 = "Melody")) || ((typeE1 = "Music")
&& (typeE2 = "Music"))) then typeE1
            else if (((typeE1 = "Melody") && (typeE2 = "int")) ||  ((typeE1 = "int")
&& (typeE2 = "Melody"))) then "Melody"
            else raise (Type_Not_Match ("The type of \"" ^ string_of_expr e1 ^ "\"
does not match the type of \"" ^ string_of_expr e2 ^ "\" to do this calculation."));

        | Add_Bin(e1,e2) ->
            let typeE1 = check_expr env e1 in
            let typeE2 = check_expr env e2 in
            if (((typeE1 = "int") && (typeE2 = "int")) || ((typeE1 = "double") &&
(typeE2 = "double")) ||
                ((typeE1 = "Melody") && (typeE2 = "Melody")) || ((typeE1 = "Music")
```

```
&& (typeE2 = "Music"))) then typeE1
            else if (((typeE1 = "Note") && (typeE2 = "Note")) || ((typeE1 =
"Melody") && (typeE2 = "Note")) ||
                ((typeE1 = "Melody") && (typeE2 = "Note"))) then "Melody"
            else if ((typeE1 = "Note") && (typeE2 = "int")) then "Note"
            else raise (Type_Not_Match ("The type of \"" ^ string_of_expr e1 ^ "\"
does not match the type of \"" ^ string_of_expr e2 ^ "\" to do this calculation."));

        | Sub(e1,e2) ->
            let typeE1 = check_expr env e1 in
            let typeE2 = check_expr env e2 in
            if (((typeE1 = "int") && (typeE2 = "int")) || ((typeE1 = "double") &&
(typeE2 = "double")) ||
                ((typeE1 = "Note") && (typeE2 = "int")) || ((typeE1 = "Melody")
&& (typeE2 = "Note"))) then typeE1
            else raise (Type_Not_Match ("The type of \"" ^ string_of_expr e1 ^ "\"
does not match the type of \"" ^ string_of_expr e2 ^ "\" to do this calculation."));


        | Less(e1,e2) | Greater(e1,e2) | Less_Equal(e1,e2) | Greater_Equal(e1,e2)
->
            let typeE1 = check_expr env e1 in
            let typeE2 = check_expr env e2 in
            if (not (typeE1 = "int")) && (not (typeE1 = "double")) then
                    raise (Type_Not_Match ("The type of \"" ^ string_of_expr e1
^ "\" was expected of type int or double."));
            if (not (typeE2 = "int")) && (not (typeE2 = "double")) then
                    raise (Type_Not_Match ("The type of \"" ^ string_of_expr e2
^ "\" was expected of type int or double."));
            if not (typeE1 = typeE2) then
                raise (Type_Not_Match ("The type of \"" ^ string_of_expr e1 ^ "\"
does not match the type of \"" ^ string_of_expr e2 ^ "\"."));
            "Boolean"

        | Equal(e1,e2) | Non_Equal(e1,e2) ->
```

```
            let typeE1 = check_expr env e1 in
            let typeE2 = check_expr env e2 in
            if not (typeE1 = typeE2) then
                raise (Type_Not_Match ("The type of \"" ^ string_of_expr e1 ^ "\"
does not match the type of \"" ^ string_of_expr e2 ^ "\"."));
            "Boolean"

        | And(e1,e2) | Or(e1,e2) ->
            let typeE1 = check_expr env e1 in
            let typeE2 = check_expr env e2 in
            if not (typeE1 = "Boolean") then
                    raise (Type_Not_Match ("The type of \"" ^ string_of_expr e1
^ "\" was expected of type boolean."));
            if not (typeE2 = "Boolean") then
                    raise (Type_Not_Match ("The type of \"" ^ string_of_expr e2
^ "\" was expected of type boolean."));
            "Boolean"

        | Expr(e1,e2,e3) ->
            (match e2 with
                ASS ->
                    let typeE1 = check_expr env e1 in
                    let typeE3 = check_expr env e3 in
                    if not (typeE1 = typeE3) then
                        raise (Type_Not_Match ("The type of \"" ^ string_of_expr
e1 ^ "\" does not match the type of \"" ^ string_of_expr e3 ^ "\"."));
                    typeE1
                | AASS ->
                    let typeE1 = check_expr env e1 in
                    let typeE3 = check_expr env e3 in
                    if (((typeE1 = "int") && (typeE3 = "int")) || ((typeE1 =
"double") && (typeE3 = "double")) ||
                        ((typeE1 = "Melody") && (typeE3 = "Melody")) || ((typeE1
= "Music") && (typeE3 = "Music")) ||
                        ((typeE1 = "Note") && (typeE3 = "int")) || ((typeE1 =
```

```
"Melody") && (typeE3 = "Note"))) then typeE1
                    else raise (Type_Not_Match ("The type of \"" ^ string_of_expr
e1 ^ "\" does not match the type of \"" ^ string_of_expr e3 ^ "\" to do this
calculation."));
                | SASS ->
                    let typeE1 = check_expr env e1 in
                    let typeE3 = check_expr env e3 in
                    if (((typeE1 = "int") && (typeE3 = "int")) || ((typeE1 =
"double") && (typeE3 = "double")) ||
                        ((typeE1 = "Note") && (typeE3 = "int"))) then typeE1
                    else raise (Type_Not_Match ("The type of \"" ^ string_of_expr
e1 ^ "\" does not match the type of \"" ^ string_of_expr e3 ^ "\" to do this
calculation."));
                | MASS ->
                    let typeE1 = check_expr env e1 in
                    let typeE3 = check_expr env e3 in
                    if (((typeE1 = "int") && (typeE3 = "int")) || ((typeE1 =
"double") && (typeE3 = "double")) ||
                        ((typeE1 = "Melody") && (typeE3 = "Melody")) || ((typeE1
= "Music") && (typeE3 = "Music")) ||
                        ((typeE1 = "Melody") && (typeE3 = "int"))) then typeE1
                    else raise (Type_Not_Match ("The type of \"" ^ string_of_expr
e1 ^ "\" does not match the type of \"" ^ string_of_expr e3 ^ "\" to do this
calculation."));)

        | Paren_Expr(e1) -> check_expr env e1

        | Array(e1,e2) ->
            let typeE1 = check_expr env e1 in
            let typeE2 = check_expr env e2 in
            if not (typeE2 = "int") then
                raise (Type_Not_Match ("The type of \"" ^ string_of_expr e2 ^ "\"
was expected of type int."));
            if not ((String.sub typeE1 0 5) = "array") then
                raise (Type_Not_Match ("The type of \"" ^ string_of_expr e1 ^ "\"
```

```
was expected of type array."));
            String.sub typeE1 6 ((String.length typeE1)-6)

        | Func_Call(e1,Some(a)) ->
            let typeE1 = check_expr env e1 in
            let typeA = String.concat "_"(List.rev_map (check_expr env) a) in
            let l = String.length typeA in
            if (((String.length typeE1) <= l) || (not ((String.sub typeE1 0 l) =
typeA))) then
                raise (Type_Not_Match ("Could not find the function \"" ^
string_of_expr e1 ^ "\" of type Function(" ^ typeA ^ ")."));
            String.sub typeE1 (l+1) ((String.length typeE1)-l-1)

        | Func_Call(e1,None) ->
            let typeE1 = check_expr env e1 in
            if not ((String.sub typeE1 0 4) = "void") then
                raise (Type_Not_Match ("Could not find the function \"" ^
string_of_expr e1 ^ "\" of type Function(void)."));
            String.sub typeE1 5 ((String.length typeE1)-5)

let rec add_val arr datatype env v =
    match v with
        VId(vid) ->
            let table = fst env in
            let scope = List.hd (snd env) in
            if Hashtbl.mem table ((string_of_int scope) ^ "_" ^ vid) then
                raise (Failure ("Varialble " ^ vid ^ " has already been
declared."));
            let typeID = ref "" in
                for i = 1 to arr do
                    typeID := !typeID ^ "array" ^ "_"
                done;
                typeID := !typeID ^ datatype;
                Hashtbl.add table ((string_of_int scope) ^ "_" ^ vid) !typeID;
                table
```

```
        | Array_Dec(vd, Some(e)) ->
            let typeE = check_expr env e in
            if not (typeE = "int") then
                raise (Type_Not_Match ("The type of \"" ^ string_of_expr e ^ "\"
was expected of type int."));
            add_val (arr + 1) datatype env vd
        | Array_Dec(vd, None) ->
            add_val (arr + 1) datatype env vd

let rec find_name v =
    match v with
    VId(vid) -> vid
    | Array_Dec(vd, e) -> find_name vd

let find_name_with_scope env v =
    let scope = List.hd (snd env) in
    (string_of_int scope) ^ "_" ^ (find_name v)

let type_of_para env d =
    match d with
        Param(t, v) ->
            let table = add_val 0 t env v in
            Hashtbl.find table (find_name_with_scope env v)


let rec add_func datatype env f =
    match f with
        Func_dec_p(s, None) ->
            let table = fst env in
            if Hashtbl.mem table s then
                raise (Failure ("Function " ^ s ^ " has already been declared."));
            let typeID = "void_void_" ^ datatype in
                Hashtbl.add table s typeID;
                table
```

```
        | Func_dec_p(s,Some(pl)) ->
            let table = fst env in
            if Hashtbl.mem table s then
                raise (Failure ("Function " ^ s ^ " has already been declared."));
            let typeID = "void_" ^ String.concat "_" (List.rev_map (type_of_para
env) pl) ^ "_" ^  datatype in
                Hashtbl.add table s typeID;
                table

let rec type_of_init env i =
    match i with
        Expr_Init(e) -> check_expr env e
        | Init(s, e) -> s
        | Func_Init(e) ->
            "array_" ^ (type_of_init env (List.hd e))

let rec check_init datatype env i =
    match i with
        Expr_Init(e) -> check_expr env e
        | Init(s, None) ->
            if ((not (s = "Note")) && (not (s = "Music")) && (not (s = "Melody"))
&& (not (s = "Track"))) then
                raise (Type_Not_Match ("The type was expected of type Note, Melody,
Track, or Music."));
            s
        | Init(s, Some(e)) ->
            let table = fst env in
            if ((not (s = "Note")) && (not (s = "Music")) && (not (s = "Melody"))
&& (not (s = "Track"))) then
                raise (Type_Not_Match ("The type was expected of type Note, Melody,
Track, or Music."));
            let e_list = List.rev_map (check_expr env) e in
            let l = List.length e_list in
            let h = List.hd e_list in
            if ((l = 1) && (h = s)) then s
```

```
            else begin
                let typeFunc = Hashtbl.find table s in
                let typeA = String.concat "_" e_list in

                if (((String.length typeA) <= (String.length typeFunc)) &&
                     ((String.sub typeFunc 0 (String.length typeA)) = typeA)) then
s
                else raise (Type_Not_Match ("The arguments do not match."));
            end
        | Func_Init(e) ->
            let eql a = (datatype = a) in
            if not (List.for_all eql (List.map (check_init datatype env) e)) then
                raise (Type_Not_Match ("The expressions in the list are expected
of type " ^ datatype ^ "."));
            "array_" ^ datatype

let rec type_of_dec_list env lst =
    let table = fst env in
    (match lst with
        Val_Decl(v) -> Hashtbl.find table (find_name_with_scope env v)
        | Assignment(v, i) -> Hashtbl.find table (find_name_with_scope env v)
        | Dec_list(d,v) -> Hashtbl.find table (find_name_with_scope env v)
        | Assign_list(d,v,i) -> Hashtbl.find table (find_name_with_scope env v))

let rec check_dec_list datatype env lst  =
    match lst with
        Val_Decl(v) ->
            if ((datatype = "Note") || (datatype = "Music") || (datatype = "Melody")
|| (datatype = "Track")) then
                raise (Need_Init ("The variable " ^ string_of_val_declarator v ^
" need to be initialized when declaring."));
            add_val 0 datatype env v
        | Assignment(v,i) ->
            let table = add_val 0 datatype env v in
            let env = (table, snd env) in
```

```
            let typeV = Hashtbl.find table (find_name_with_scope env v) in
            let typeI = check_init datatype env i in
            if not (typeI = typeV) then
                raise (Type_Not_Match ("The type of \"" ^ string_of_init i ^ "\"
was expected of type " ^ datatype ^ "."));
            table

        | Dec_list(d,v) ->
            if ( (datatype = "Note") || (datatype = "Music") || (datatype =
"Melody") || (datatype = "Track")) then
                raise (Need_Init ("The variable " ^ string_of_val_declarator v ^
" need to be initialized when declaring."));
            let env = (check_dec_list datatype env d, snd env) in
                add_val 0 datatype env v
        | Assign_list(d,v,i) ->
            let env = (check_dec_list datatype env d, snd env) in
            let typeI = check_init datatype env i in
            if not (typeI = datatype) then
                raise (Type_Not_Match ("The type of \"" ^ string_of_init i ^ "\"
was expected of type " ^ datatype ^ "."));
            add_val 0 datatype env v

let rec check_stmt env stmt =
    match stmt with
        Stmt(None) -> fst env
        | Stmt(Some(e)) ->
            let typeID = check_expr env e in
                fst env
        | Dec(datatype,d) -> check_dec_list datatype env d
        | Bre_Stmt(None) -> fst env
        | Bre_Stmt(Some(stmts)) ->
            let new_scope = inc_scope() in
            let env = (fst env, new_scope :: (snd env))    in
                call_list check_stmt env stmts
```

```
        | If(e,stmt) ->
            let typeID = check_expr env e in
            if not (typeID = "Boolean") then
                raise (Type_Not_Match ("The type of \"" ^ string_of_expr e ^ "\"
was expected of type boolean."));
            check_stmt env stmt
        | If_else(e,stmt1,stmt2) ->
            let typeID = check_expr env e in
            if not (typeID = "Boolean") then
                raise (Type_Not_Match ("The type of \"" ^ string_of_expr e ^ "\"
was expected of type boolean."));
            let table = check_stmt env stmt1 in
            let env = (table, snd env) in
                check_stmt env stmt2

        | While(e,stmt) ->
            let typeID = check_expr env e in
            if not (typeID = "Boolean") then
                raise (Type_Not_Match ("The type of \"" ^ string_of_expr e ^ "\"
was expected of type boolean."));
            let table = fst env in
            let previous_loop = Hashtbl.find table "current_loop" in
            Hashtbl.replace table "current_loop" "true";
            let env = (table, snd env) in
            let table = check_stmt env stmt in
            Hashtbl.replace table "current_loop" previous_loop;
            table

        | For_complete(stmt1,stmt2,e,stmt3) ->
            let table = check_stmt env stmt1 in
            let env = (table, snd env) in
            let table = check_stmt env stmt2 in
            let env = (table, snd env) in
            let typeID = check_expr env e in
            let previous_loop = Hashtbl.find table "current_loop" in
```

```
            Hashtbl.replace table "current_loop" "true";
            let env = (table, snd env) in
            let table = check_stmt env stmt3 in
            Hashtbl.replace table "current_loop" previous_loop;
            table

        | For_part(stmt1,stmt2,stmt3) ->
            let table = check_stmt env stmt1 in
            let env = (table, snd env) in
            let table = check_stmt env stmt2 in
            let previous_loop = Hashtbl.find table "current_loop" in
            Hashtbl.replace table "current_loop" "true";
            let env = (table, snd env) in
            let table = check_stmt env stmt3 in
            Hashtbl.replace table "current_loop" previous_loop;
            table

        | CONTINUE ->
            let table = fst env in
            let loop = Hashtbl.find table "current_loop" in
            if not (loop = "true") then
                raise (Type_Not_Match ("There is no loop to match the
\"continue\"."));
            table
        | BREAK ->
            let table = fst env in
            let loop = Hashtbl.find table "current_loop" in
            if not (loop = "true") then
                raise (Type_Not_Match ("There is no loop to match the
\"break\"."));
            table

        | RETURN ->
            let table = fst env in
            let typeID = Hashtbl.find table "current_function_type" in
```

```
            if not (typeID = "void") then
                raise (Type_Not_Match ("The type of return value is expected of
type " ^ typeID ^ "."));
            table
        | Return(e) ->
            let typeE = check_expr env e in
            let table = fst env in
            let typeID = Hashtbl.find table "current_function_type" in
            if not (typeID = typeE) then
                raise (Type_Not_Match ("The type of return value is expected of
type " ^ typeID ^ "."));
            table

let check_func_def env func =
    match func with
    Func_Def(s, d, stmts) ->
        let new_scope = inc_scope() in
        let env = (fst env, new_scope :: (snd env))    in
        let table = add_func s env d in
        Hashtbl.replace table "current_function_type" s;
        let env = (table, snd env)    in
            call_list check_stmt env stmts

let print key value =
    print_string (key ^ " " ^ value ^ "\n")

let check_program (stmts, funcs) =
    let table = base_table in
    let env = (table, [0]) in
    let table = call_list check_stmt env stmts in
    let table = call_list check_func_def env (List.rev funcs) in
    if not (Hashtbl.mem table "main") then
        raise (No_Main ("There is no \"main\" function in this program."));
    let typeM = Hashtbl.find table "main" in
    if not (typeM = "void_array_String_void") then
```

```
        raise (No_Main ("The argument of \"main\" function are expected to be
\"string[]\", and return void."));
    table
```

JavaCode Generator:

```
open Printf
 open Checker
 open Ast

 let java_scope = ref 0
 let gen_inc_scope (u:unit) =
     java_scope := !java_scope + 1;
     !java_scope

 let rec string_of_expr env expr =
     match expr with
     | Id(s) -> s
     | IntCon(c) -> string_of_int c
     | DoubleCon(d) -> string_of_float d
     | BoolCon(b) -> string_of_bool b
     | StrCon(s) -> s
     | InsCon(s) | PitCon(s) -> "C." ^ s
     | Dot_Expr(e,s) -> string_of_expr env e ^ "." ^ s
     | Self_Add(e) -> string_of_expr env e ^ "++"
     | Self_Sub(e) -> string_of_expr env e ^ "--"
     | Self_Add2(e) -> "++" ^ string_of_expr env e
     | Self_Sub2(e) -> "--" ^ string_of_expr env e
     | Uminus(e) -> "-" ^ string_of_expr env e
     | Not_Unary(e) -> "!" ^ string_of_expr env e

     | Multi(e1,e2) -> (
         let typeE1 = check_expr env e1 in
         let typeE2 = check_expr env e2 in
```

```
        match (typeE1,typeE2) with
        | ("Melody","Melody") -> "PublicFunction.multiplyMelody("^
string_of_expr env e1 ^ ", " ^ string_of_expr env e2^")"
        | ("Music","Music") -> "PublicFunction.multiplyMusic("^ string_of_expr
env e1 ^ ", " ^ string_of_expr env e2^")"
        | ("Melody","int") | ("int","Melody") -> "PublicFunction.multiplyInt("^
string_of_expr env e1 ^ ", " ^ string_of_expr env e2^")"
        | _ -> "(" ^ string_of_expr env e1 ^ ")" ^ " * " ^ "(" ^ string_of_expr
env e2 ^ ")"
    )

    | Add_Bin(e1,e2) -> (
        let typeE1 = check_expr env e1 in
        let typeE2 = check_expr env e2 in
        match (typeE1,typeE2) with
        | ("Melody","Melody") -> "PublicFunction.addMelody(" ^ string_of_expr
env e1 ^ ", " ^ string_of_expr env e2^")"
        | ("Music","Music") ->    "PublicFunction.addMusic(" ^ string_of_expr
env e1 ^ ", " ^ string_of_expr env e2^")"
        | ("Note","Note") -> "PublicFunction.addNote("^ string_of_expr env e1 ^
", " ^ string_of_expr env e2^")"
        | ("Note","int") | ("int","Note") -> "PublicFunction.plus(" ^
string_of_expr env e1 ^ ", " ^ string_of_expr env e2^")"
        | ("Melody","Note") | ("Note", "Meldoy") -> "PublicFunction.addNote(" ^
string_of_expr env e1 ^ ", " ^ string_of_expr env e2^")"
        | _ -> string_of_expr env e1 ^ " + " ^ string_of_expr env e2
    )


    | Sub(e1,e2) -> (
        let typeE1 = check_expr env e1 in
        let typeE2 = check_expr env e2 in
        match (typeE1,typeE2) with
        | ("Note","int") -> "PublicFunction.minus(" ^ string_of_expr env e1 ^ ",
" ^ string_of_expr env e2^")"
```

```
        | _ -> string_of_expr env e1 ^ " - " ^ string_of_expr env e2
    )


    | Less(e1,e2) -> string_of_expr env e1 ^ " < " ^ string_of_expr env e2
    | Greater(e1,e2) -> string_of_expr env e1 ^ " > " ^ string_of_expr env e2
    | Less_Equal(e1,e2) -> string_of_expr env e1 ^ " <= " ^ string_of_expr env e2
    | Greater_Equal(e1,e2) -> string_of_expr env e1 ^ " >= " ^ string_of_expr env
e2

    | Equal(e1,e2) -> string_of_expr env e1 ^ " == " ^ string_of_expr env e2
    | Non_Equal(e1,e2) -> string_of_expr env e1 ^ " != " ^ string_of_expr env e2

    | And(e1,e2) -> string_of_expr env e1 ^ " && " ^ string_of_expr env e2

    | Or(e1,e2) -> string_of_expr env e1 ^ " || " ^ string_of_expr env e2

    | Expr(e1,e2,e3) -> (
        match e2 with
            | AASS ->
                let typeE1 = check_expr env e1 in
                let typeE3 = check_expr env e3 in
                (match (typeE1, typeE3) with
                    ("Melody", "Melody") -> string_of_expr env e1 ^ "." ^
"addMelody(" ^ string_of_expr env e3 ^ ")"
                    | ("Music", "Music") -> string_of_expr env e1 ^ "." ^
"addMusic(" ^ string_of_expr env e3 ^ ")"
                    | ("Note", "int") -> string_of_expr env e1 ^ "." ^ "plus("
^ string_of_expr env e3 ^ ")"
                    | ("Melody", "Note") -> string_of_expr env e1 ^ "." ^
"addNote(" ^ string_of_expr env e3 ^ ")"
                    | _ -> string_of_expr env e1 ^ " += " ^ string_of_expr env
e3
                )
```

```
            | SASS ->
                let typeE1 = check_expr env e1 in
                let typeE3 = check_expr env e3 in
                if(typeE1 = "Note" && typeE3 = "int") then
                    string_of_expr env e1 ^ "." ^ "minus(" ^ string_of_expr env
e3 ^ ")"
                else  string_of_expr env e1 ^ " -= " ^ string_of_expr env e3

            | MASS ->
                let typeE1 = check_expr env e1 in
                let typeE3 = check_expr env e3 in
                (match (typeE1, typeE3) with
                    ("Melody", "Melody") -> string_of_expr env e1 ^ "." ^
"multiplyMelody(" ^ string_of_expr env e3 ^ ")"
                    | ("Music", "Music") -> string_of_expr env e1 ^ "." ^
"multiplyMusic(" ^ string_of_expr env e3 ^ ")"
                    | ("Melody", "int") -> string_of_expr env e1 ^ "." ^
"multiplyInt(" ^ string_of_expr env e3 ^ ")"
                    | _ -> string_of_expr env e1 ^ " *= " ^ string_of_expr env
e3
                )
            | ASS -> string_of_expr env e1 ^ " = " ^ string_of_expr env e3

    )
    | Paren_Expr(e1) -> string_of_expr env e1
    | Array(e1,e2) ->
        let typeE = check_expr env e1 in
        if ((typeE = "array_Track") || (typeE = "array_Note")) then
            string_of_expr env e1 ^ ".get(" ^ string_of_expr env e2 ^ ")"
        else string_of_expr env e1 ^ "[" ^ string_of_expr env e2 ^ "]"
    | Func_Call(e1,Some(a)) ->
        let stringE = string_of_expr env e1 in
        (match stringE with
            | "print" -> "PublicFunction.print(" ^ String.concat ", "
(List.rev_map (string_of_expr env) a) ^ ")"
```

```
            | "read" -> "PublicFunction.read(" ^ String.concat ", " (List.rev_map
(string_of_expr env) a) ^ ")"
            | "write" -> "PublicFunction.write(" ^ String.concat ", "
(List.rev_map (string_of_expr env) a) ^ ")"
            | "setNoteDefault" -> "PublicFunction.setNoteDefault(" ^
String.concat ", " (List.rev_map (string_of_expr env) a) ^ ")"
            | "changeToMillisecond" -> "PublicFunction.changeToMillisecond("^
String.concat ", " (List.rev_map (string_of_expr env) a) ^ ")"
            | "printTime" -> "PublicFunction.printTime("^ String.concat ", "
(List.rev_map (string_of_expr env) a) ^ ")"
            | "sizeOf" -> "PublicFunction.sizeOf("^ String.concat ", "
(List.rev_map (string_of_expr env) a) ^ ")"
            | _ -> string_of_expr env e1 ^ "(" ^ String.concat ", "(List.rev_map
(string_of_expr env) a) ^ ")" (* linkdown(ip,port)*)
        )

    | Func_Call(e1,None) -> string_of_expr env e1 ^ "(" ^ ")" (*linkdown()*)



let rec string_of_val_declarator env decl =
    match decl with
    | VId(s) -> s (**)
    | Array_Dec(d,Some(e)) -> string_of_val_declarator env d ^ "[" ^
string_of_expr env e ^ "]" (* a[4] *)
    | Array_Dec(d,None) -> string_of_val_declarator env d ^ "[" ^ "]" (* a[] *)

let rec string_of_init env i =
    match i with
    | Expr_Init(e) -> string_of_expr env e
    | Init(s,None)  -> (
        match s with
        | "Note" -> "new Note()"
        | "Melody" -> "new Melody()"
        | "Music" -> "new Music()"
```

```
        | "Track" -> "new Track()"
        | _ -> s
    )
    | Init(s, Some(e)) -> (
        match s with
        | "Note" -> "new Note(" ^ String.concat ", " (List.rev_map (string_of_expr
env) e) ^ ")"
        | "Melody" -> "new Melody(" ^ String.concat ", " (List.rev_map
(string_of_expr env) e) ^ ")"
        | "Music" -> "new Music(" ^ String.concat ", " (List.rev_map
(string_of_expr env) e) ^ ")"
        | "Track" -> "new Track(" ^ String.concat ", " (List.rev_map
(string_of_expr env) e) ^ ")"
        | _ -> s
    )
    | Func_Init(e)  -> (
        let typeE = type_of_init env (List.hd e) in
        match typeE with
        | "Track" -> "new ArrayList<Track>(Arrays.asList("^ String.concat ", "
(List.rev_map (string_of_init env) e)^"))"
        | "Note" -> "new ArrayList<Note>(Arrays.asList("^ String.concat ", "
(List.rev_map (string_of_init env) e)^"))"
        |  _ -> "{" ^ String.concat ", " (List.rev_map (string_of_init env) e)
^ "}"
    )

let rec string_of_dec_list env flag decl =
    match decl with
    | Val_Decl(v) -> string_of_val_declarator env v
    | Assignment(v,i) ->
        if flag then
            let name = find_name v in
            name ^ " = " ^ string_of_init env i
        else
            (string_of_val_declarator env v) ^ " = " ^ string_of_init env i
```

```
    | Dec_list(d,v) -> string_of_dec_list env flag d ^ ", " ^
string_of_val_declarator env v
    | Assign_list(d,v,i) ->
        if flag then
            let name = find_name v in
            string_of_dec_list env flag d ^ ", " ^ name ^ " = " ^ string_of_init
env i
        else
            string_of_dec_list env flag d ^ ", " ^ string_of_val_declarator env
v ^ " = " ^ string_of_init env i




let rec string_of_stmt env stmt =
    match stmt with
    | Stmt(None) -> ";"
    | Stmt(Some(e)) -> string_of_expr env e ^ ";\n"
    | Bre_Stmt(None) -> "{\n}"
    | Bre_Stmt(Some(stmts)) ->
        let new_scope = gen_inc_scope() in
        let env = (fst env, new_scope :: (snd env))    in
        "{\n  " ^ String.concat "\n" (List.map (string_of_stmt env) stmts) ^ "\n}"
    | Dec(datatype,d) -> (
        let typed = type_of_dec_list env d in
        match typed with
        | "array_Track"-> "List<Track> " ^ string_of_dec_list env true d ^ ";"
        | "array_Note" -> "List<Note> " ^ string_of_dec_list env true d ^ ";"
        | _ -> datatype ^ " " ^ string_of_dec_list env false d ^ ";"

    )
    | If(e,stmt) -> "if " ^ "(" ^ string_of_expr env e ^ ")" ^ string_of_stmt env
stmt
    | If_else(e,stmt1,stmt2) ->
```

```
        let strE = string_of_expr env e in
        let strIF = string_of_stmt env stmt1 in
        let strEL = string_of_stmt env stmt2 in
        "if" ^ "(" ^ strE ^ ")" ^ strIF ^ "else" ^ strEL
    | While(e,stmt) -> "while" ^ "(" ^ string_of_expr env e ^ ")" ^ string_of_stmt
env stmt
    | For_complete(stmt1,stmt2,e,stmt3) -> "for" ^ "(" ^ string_of_stmt env stmt1
^ string_of_stmt env stmt2 ^ string_of_expr env e ^ ")\n" ^ string_of_stmt env stmt3
    | For_part(stmt1,stmt2,stmt3) -> "for" ^ "(" ^ string_of_stmt env stmt1 ^
string_of_stmt env stmt2 ^ ")" ^ string_of_stmt env stmt3
    | CONTINUE -> "continue;"
    | BREAK -> "break;"
    | RETURN -> "return;"
    | Return(e) -> "return " ^ string_of_expr env e ^ ";"


let rec string_of_param env para =
    match para with
        Param(s,v) ->  s ^ " " ^ string_of_val_declarator env v


let rec string_of_func_declarator env funcDec =
    match funcDec with
        Func_dec_p(s,Some(pl)) -> s ^ "(" ^ String.concat "," (List.rev_map
(string_of_param env) pl) ^ ")"
        | Func_dec_p(s,None) -> s ^ "(" ^ " " ^ ")"

let rec string_of_func_def env funcDef =
    let new_scope = gen_inc_scope() in
    let env = (fst env, new_scope :: (snd env))    in
    (match funcDef with
        Func_Def(s, d, stmts) ->
            "public static " ^ s ^ " " ^ string_of_func_declarator env d ^ "{\n"
^ String.concat "\n" (List.map (string_of_stmt env) stmts) ^ "\n}\n")
```

```
let string_of_program env (stmts, funcs) =
  String.concat "\n" (List.rev_map (string_of_stmt env) stmts) ^ "\n" ^
  String.concat "\n" (List.map (string_of_func_def env) (List.rev funcs)) ^ "\n"

let  writeToFile fileName progString =
  let file = open_out ( fileName ^ ".java") in
    let content = fprintf file "%s"  progString in
        "Write finished!"

let gen_program fileName table prog =
    let env = (table, [0]) in
    let programString = string_of_program env (fst prog, snd prog) in
    let out = sprintf
        "import java.io.*;\nimport java.util.*;\nimport definition.*;\nimport
function.PublicFunction;\n\npublic class %s\n{\n%s\n}" fileName programString in
    writeToFile fileName out
```

C.java

```
package definition;

public class C {
    public static final int C1 = 24;
    public static final int D1 = 26;
    public static final int E1 = 28;
    public static final int F1 = 29;
    public static final int G1 = 31;
    public static final int A1 = 33;
    public static final int B1 = 35;

    public static final int C2 = 36;
    public static final int D2 = 38;
```

```
public static final int E2 = 40;
public static final int F2 = 41;
public static final int G2 = 43;
public static final int A2 = 45;
public static final int B2 = 47;

public static final int C3 = 48;
public static final int D3 = 50;
public static final int E3 = 52;
public static final int F3 = 53;
public static final int G3 = 55;
public static final int A3 = 57;
public static final int B3 = 59;

public static final int C4 = 60;
public static final int D4 = 62;
public static final int E4 = 64;
public static final int F4 = 65;
public static final int G4 = 67;
public static final int A4 = 69;
public static final int B4 = 71;

public static final int C5 = 72;
public static final int D5 = 74;
public static final int E5 = 76;
public static final int F5 = 77;
public static final int G5 = 79;
public static final int A5 = 81;
public static final int B5 = 83;

public static final int C6 = 84;
public static final int D6 = 86;
public static final int E6 = 88;
public static final int F6 = 89;
public static final int G6 = 91;
```

```
public static final int A6 = 93;
public static final int B6 = 95;

public static final int C7 = 96;
public static final int D7 = 98;
public static final int E7 = 100;
public static final int F7 = 101;
public static final int G7 = 103;
public static final int A7 = 105;
public static final int B7 = 107;

public static final int C8 = 108;
public static final int D8 = 110;
public static final int E8 = 112;
public static final int F8 = 113;
public static final int G8 = 115;
public static final int A8 = 117;
public static final int B8 = 119;

public static final int C9 = 120;
public static final int D9 = 122;
public static final int E9 = 124;
public static final int F9 = 125;
public static final int G9 = 127;
public static final int A9 = 129;
public static final int B9 = 131;

public static final int PIANO = 0;
public static final int VIOLIN = 40;
public static final int ACCORDION = 21;
public static final int FLUTE = 73;
public static final int OBOE = 68;
public static final int TRUMPET = 56;

public static int NOTE_DEFAULT_PITCH = C4;
```

```
    public static int NOTE_DEFAULT_STARTTIME = 0;
    public static int NOTE_DEFAULT_DURATION = 500;
    public static int NOTE_DEFAULT_STRENGTH = 100;
}
```

Note.java

```
package definition;
import java.util.*;
public class Note {

    private int m_pitch;
    private int m_duration;
    private int m_startTime;
    private int m_strength;

    public Note()
    {
        m_pitch = C.NOTE_DEFAULT_PITCH;
        m_duration = C.NOTE_DEFAULT_DURATION;
        m_startTime = C.NOTE_DEFAULT_STARTTIME;
        m_strength = C.NOTE_DEFAULT_STRENGTH;
        C.NOTE_DEFAULT_STARTTIME += C.NOTE_DEFAULT_DURATION;
    }

    public Note(int pitch)
    {
        m_pitch = pitch;
        m_duration = C.NOTE_DEFAULT_DURATION;
        m_startTime = C.NOTE_DEFAULT_STARTTIME;
        m_strength = C.NOTE_DEFAULT_STRENGTH;
```

```
        C.NOTE_DEFAULT_STARTTIME += C.NOTE_DEFAULT_DURATION;
    }

    public Note(int pitch, int duration)
    {
        m_pitch = pitch;
        m_duration = duration;
        m_startTime = C.NOTE_DEFAULT_STARTTIME;
        m_strength = C.NOTE_DEFAULT_STRENGTH;
        C.NOTE_DEFAULT_STARTTIME += duration;
    }

    public Note(int pitch, int duration, int startTime)
    {
        m_pitch = pitch;
        m_duration = duration;
        m_startTime = startTime;
        m_strength = C.NOTE_DEFAULT_STRENGTH;
    }

    public Note(int pitch, int duration, int startTime, int strength)
    {
        m_pitch = pitch;
        m_duration = duration;
        m_startTime = startTime;
        m_strength = strength;
    }

    public Note(Note note)
    {
        m_pitch = note.getPitch();
        m_duration = note.getDuration();
        m_startTime = note.getStartTime();
        m_strength = note.getStrength();
    }
```

```
public void setDuration(int duration)
{
    m_duration = duration;
}

public int getDuration()
{
    return m_duration;
}

public void setPitch(int pitch)
{
    m_pitch = pitch;
}

public int getPitch()
{
    return m_pitch;
}

public void setStrength(int strength)
{
    m_strength = strength;
}

public int getStrength()
{
    return m_strength;
}

public void setStartTime(int startTime)
{
    m_startTime = startTime;
}
```

```
    public int getStartTime()
    {
        return m_startTime;
    }

    public Note plus(int val){
        return new Note(m_pitch+val, m_duration, m_startTime, m_strength);
    }

    public Note minus(int val){

        return new Note(m_pitch-val, m_duration, m_startTime, m_strength);
    }

}
```

Melody.java

```
package definition;
 import java.util.*;
 import function.*;

 public class Melody {

    private List<Note> m_noteList;

    public Melody(){

        m_noteList = new ArrayList<Note>();
    }
```

```
public Melody(List<Note> noteList){

    m_noteList = new ArrayList<Note>(noteList);
}

public Melody(Melody melody){

    if(m_noteList == null)
        m_noteList = new ArrayList<Note>();

    m_noteList = new ArrayList<Note>(melody.getNoteList());
}

public List<Note> getNoteList(){

    return m_noteList;
}

public void setNoteList(List<Note> noteList){

    m_noteList = noteList;
}

public Melody subMelody(int startPoint, int endPoint){

    List<Note> newNoteList = new ArrayList<Note>();

    if((startPoint < 0) || (endPoint >= m_noteList.size()))
            throw new NullPointerException(); // extends exception

    for(int i=startPoint;i<=endPoint;i++)
        newNoteList.add(m_noteList.get(i));


    Melody subMelody = new Melody(newNoteList);
```

```
        return subMelody;
    }

    public void addNote(Note note){

        if(m_noteList == null)
            m_noteList = new ArrayList<Note>();

        m_noteList.add(note);

    }

    public void deleteNote(int index){

        if(index >= m_noteList.size())
            throw new NullPointerException();

        m_noteList.remove(index);
    }

    public Note getNote(int index){

        return m_noteList.get(index);
    }

    public int getLength(){

        return m_noteList.size();
    }

    public int getTimeLength(){

        int maxEnd = 0;
        for(Note note : m_noteList)
        {
```

```
            int endtime = note.getStartTime() + note.getDuration();
            if(endtime > maxEnd)
                maxEnd = endtime;
        }
        return maxEnd;
    }

    //melody  = melody + melody
    public void addMelody(Melody melody_2)
    {
        List<Note> noteList_2 = melody_2.getNoteList();

        if((noteList_2 == null) ||  (m_noteList == null))
            throw new NullPointerException();

        int listSize = m_noteList.size();
        Note lastNote = m_noteList.get(listSize-1);
        int startTime = lastNote.getStartTime() + lastNote.getDuration();

        for(Note note: noteList_2)
        {
            Note newNote = new Note(note);
            newNote.setStartTime(startTime);
            m_noteList.add(newNote);
            startTime = startTime + note.getDuration();
        }

    }

    //multiplyMelody
    public void multiplyMelody(Melody melody_2)
    {
        List<Note> noteList_2 = melody_2.getNoteList();

        if((m_noteList == null) || (noteList_2 == null))
```

```
            throw new NullPointerException();

        m_noteList.addAll(noteList_2);
    }

    //multiplyInt
    public void multiplyInt(int time){
        Melody newMelody = new Melody(this);

        for(int i=0;i<time-1;i++){
            addMelody(newMelody);

        }

    }



}
```

Track . java

```
 package definition;
public class Track {
    private int t_timbre;
    private Melody t_melody;

    public Track(){
        t_melody = new Melody();
        // Default timbre is set to PIANO
        t_timbre = 0;
```

```
    }
    public Track(Melody melody){
        t_melody = melody;
        // Default is set to PIANO
        t_timbre = 0;
    }

    public Track(Melody melody, int timbre){
        t_melody = melody;
        t_timbre = timbre;
    }

    public Track(Track track){
        t_melody = new Melody(track.getMelody());
        t_timbre = track.getTimbre();
    }

    public void setTimbre(int timbre){
        t_timbre = timbre;
    }

    public void insertMelody(int startTime, Melody melody){

        if(startTime < 0)
            throw new IllegalArgumentException();

        for(int i = 0; i < melody.getLength(); i++){
            Note note = melody.getNote(0);
            int tmp_startTime = note.getStartTime() + startTime;
            note.setStartTime(tmp_startTime);
            t_melody.addNote(note);
        }
    }

    public Melody getMelody(){
```

```
        return t_melody;
    }

    public void setMelody(Melody melody)
    {
        t_melody = melody;
    }

    public int getTimbre(){
        return t_timbre;
    }

    public int getLength(){
        return t_melody.getLength();
    }
}
```

Music .java

```
package definition;
 import java.util.*;


 public class Music {

    private List<Track> m_trackList;

    public Music(){
        m_trackList = new ArrayList<Track>();
    }

    public Music(List<Track> trackList){
        m_trackList = trackList;
```

```
    }

    public Music(Music music){
        m_trackList = new ArrayList<Track>(music.getTracks());
    }

    public void insertTrack(Track track){
        m_trackList.add(track);
    }

    public Track getTrack(int index){
        if(index >= m_trackList.size())
            throw new NullPointerException();

        return m_trackList.get(index);
    }

    public List<Track> getTracks(){
        return m_trackList;
    }

    public int getNumberOfTracks(){
        return m_trackList.size();
    }

    public int getTimeLength(){
        int maxTimeLength = 0;
        for(Track track : m_trackList)
            if(track.getMelody().getTimeLength() > maxTimeLength)
                maxTimeLength = track.getMelody().getTimeLength();

        return maxTimeLength;
    }

    public void addMusic(Music music)// music played later
```

```
{
    List<Track> trackList = music.getTracks();

    if(trackList == null)
            throw new NullPointerException();

    int maxEndTime = getTimeLength();

    for(Track track : trackList)
    {
        Melody melody = track.getMelody();
        if(melody == null)
            throw new NullPointerException();

        List<Note> noteLists = melody.getNoteList();
        if(noteLists == null)
            throw new NullPointerException();

        int startTime = maxEndTime;
        for(Note note: noteLists)
        {
            note.setStartTime(startTime);
            startTime = startTime + note.getDuration();
        }
        melody.setNoteList(noteLists);
        track.setMelody(melody);
    }
    m_trackList.addAll(trackList);

    for(Track track: m_trackList){
        Melody melody = track.getMelody();
        System.out.println("");
        for(Note note: melody.getNoteList())
            System.out.println(note.getStartTime());
    }
```

```
    }

    public void multiplyMusic(Music music)
    {
        List<Track> trackList = music.getTracks();

        if((trackList == null) )
                throw new NullPointerException();

        for(Track track : trackList)
        {
            Melody melody = track.getMelody();
            if(melody == null)
                throw new NullPointerException();

            List<Note> noteLists = melody.getNoteList();
            if(noteLists == null)
                throw new NullPointerException();

        }
        m_trackList.addAll(trackList);
    }

}
```

Publicfunction.java

```
package function;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
```

```
import java.util.List;

import com.leff.midi.MidiFile;
import com.leff.midi.MidiTrack;
import com.leff.midi.event.MidiEvent;
import com.leff.midi.event.NoteOff;
import com.leff.midi.event.NoteOn;
import com.leff.midi.event.ProgramChange;

import definition.*;

public class PublicFunction {
    public static void setNoteDefault(int pitch, int duration, int startTime,
            int strength){

        C.NOTE_DEFAULT_PITCH = pitch;
        C.NOTE_DEFAULT_STARTTIME = startTime;
        C.NOTE_DEFAULT_DURATION = duration;
        C.NOTE_DEFAULT_STRENGTH = strength;
    }

    public static int changeToMillisecond(int minute, int second,
            int millisecond){
        return minute*60*1000 + second*1000 + millisecond;
    }

    public static void printTime(int millisecond){
        int ms = millisecond % 1000;
        millisecond /= 1000;
        int s = millisecond % 60;
        millisecond /= 60;
        int min = millisecond;
        System.out.println(min+"/"+s+"/"+ms);
    }
```

```
public static Music read(String path){
    // System.out.println("\n/********read*******/");
    // 1. Open up a MIDI file
    MidiFile mf = null;
    File input = new File(path);

    try{
        mf = new MidiFile(input);
    } catch(IOException e){
        System.err.println("Error parsing MIDI file:");
        e.printStackTrace();
        return null;
    }

    Music mus = new Music();
    // System.out.println(mf.getTrackCount());
    for(int i = 0; i < mf.getTrackCount(); i++){
     MidiTrack T = mf.getTracks().get(i);
     Iterator<MidiEvent> it = T.getEvents().iterator();

     Melody mel = new Melody();
     int timbre = -1;
     while(it.hasNext()){
            MidiEvent E = it.next();
            if(E.getClass().equals(ProgramChange.class)){
             ProgramChange p = (ProgramChange) E;
             timbre = p.getProgramNumber();
            }
            if(E.getClass().equals(NoteOn.class)){
             NoteOn sn = (NoteOn) E;
             NoteOn en = (NoteOn) it.next();

                int pitch = sn.getNoteValue();
                int duration = (int)(en.getTick()-sn.getTick());
                int startTime = (int)sn.getTick();
```

```
                    int strength = sn.getVelocity();

                    // System.out.println(timbre + ", " + pitch + ", " +
strength + ", "
                //      + startTime + ", " + duration);
                    mel.addNote(new Note(pitch, duration, startTime,
strength));
                }
            }
        Track tra = new Track(mel, timbre);
        mus.insertTrack(tra);
        }

        return mus;
    }

    public static void write(Music music, String outputpath){

        // System.out.println("\n/********write*******/");

        List<Track> tracklist = music.getTracks();
        int channel = 0;
        ArrayList<MidiTrack> midiTrackList = new ArrayList<MidiTrack>();
        for(Track track: tracklist){
            MidiTrack miditrack = new MidiTrack();
            int timbre = track.getTimbre();
            MidiEvent midievent = new ProgramChange(0, channel, timbre);
            miditrack.insertEvent(midievent);

            Melody melody = track.getMelody();
            List<Note> notelist = melody.getNoteList();
            int time = 1;
            //int tempo = 120;
            for(Note note: notelist){
                int pitch = note.getPitch();
```

```
                int duration = note.getDuration();
                int startTime = note.getStartTime();
                int strength = note.getStrength();
                /*
                 System.out.println(channel + ", " + pitch + ", " + strength
+ ", "
                        + startTime + ", " + duration);
                        */
                 if((startTime == 2000)&& (time == 2))
                     break;
                 else if(startTime == 2000)
                     time = time + 1;

    miditrack.insertNote(channel,pitch,strength,startTime,duration);
            }
            channel++;
            midiTrackList.add(miditrack);
        }

        // System.out.println(midiTrackList.size());
        MidiFile midi = new MidiFile(MidiFile.DEFAULT_RESOLUTION,
midiTrackList);

        // 4. Write the MIDI data to a file
        File output = new File(outputpath);
        try {
            midi.writeToFile(output);
        }
       catch(IOException e) {
            System.err.println(e);
       }
       // System.out.println("haha");
    }

    public static void print(String str){
```

```
        System.out.print(str);
    }

    public static int sizeof(Note notes[]){
        return notes.length;
    }

    // note = note + 2
    public static Note plus(Note note, int val){
        int pitch = note.getPitch();
        int duration = note.getDuration();
        int startTime = note.getStartTime();
        int strength = note.getStrength();

        return new Note(pitch+val, duration, startTime, strength);
    }

    // note = 2 + note
    public static Note plus(int val, Note note){
        int pitch = note.getPitch();
        int duration = note.getDuration();
        int startTime = note.getStartTime();
        int strength = note.getStrength();

        return new Note(pitch+val, duration, startTime, strength);
    }

    // note = note - 2
    public static Note minus(Note note, int val){
        int pitch = note.getPitch();
        int duration = note.getDuration();
        int startTime = note.getStartTime();
        int strength = note.getStrength();

        return new Note(pitch-val, duration, startTime, strength);
```

```
}

// Melody = note + note
public static Melody addNote(Note note_1, Note note_2){

    List<Note> noteList = new ArrayList<Note>();
    noteList.add(note_1);
    noteList.add(note_2);

    Melody melody = new Melody(noteList);

    return melody;
}

//melody  = melody + melody
public static Melody addMelody(Melody melody_1, Melody melody_2)
{
    List<Note> noteList_1 = melody_1.getNoteList();
    List<Note> noteList_2 = melody_2.getNoteList();

    List<Note> newNoteList = new ArrayList<Note>();

    if((noteList_2 == null) ||  (noteList_1 == null))
        throw new NullPointerException();

    int listSize = noteList_1.size();
    Note lastNote = noteList_1.get(listSize-1);
    int startTime = lastNote.getStartTime() + lastNote.getDuration();

    newNoteList.addAll(noteList_1);
    for(Note note: noteList_2)
    {
        Note tmp = new Note(note);
        tmp.setStartTime(startTime);
        newNoteList.add(tmp);
```

```
            startTime = startTime + note.getDuration();
        }

        Melody melody = new Melody(newNoteList);
        return melody;
    }

    //melody  = melody * melody
    public static Melody multiplyMelody(Melody melody_1, Melody melody_2)
    {
        List<Note> noteList_1 = melody_1.getNoteList();
        List<Note> noteList_2 = melody_2.getNoteList();

        if((noteList_1 == null) || (noteList_2 == null))
            throw new NullPointerException();

        List<Note> newNoteList = new ArrayList<Note>(noteList_1);

        newNoteList.addAll(noteList_2);
        Melody melody = new Melody(newNoteList);
        return melody;
    }

    //melody  = melody + note
    public static Melody addNote(Melody melody, Note note){

        List<Note> noteList = melody.getNoteList();

        if(noteList == null)
            noteList = new ArrayList<Note>();

        List<Note> newNoteList = new ArrayList<Note>();
        newNoteList.addAll(noteList);
        newNoteList.add(note);
```

```
        Melody newMelody = new Melody(newNoteList);
        return newMelody;
    }

    //melody  = note + melody
    public static Melody addNote(Note note, Melody melody){

        List<Note> noteList = melody.getNoteList();

        if(noteList == null)
            noteList = new ArrayList<Note>();

        List<Note> newNoteList = new ArrayList<Note>();
        newNoteList.addAll(noteList);
        newNoteList.add(note);

        Melody newMelody = new Melody(newNoteList);
        return newMelody;
    }

    //melody = melody * 3
    public static Melody multiplyInt(Melody melody, int time){

         Melody newMelody = new Melody(melody);
            for(int i=0;i<time-1;i++){
                System.out.println(newMelody.getLength());
                newMelody = addMelody(newMelody,melody);
            }
            return newMelody;
    }

    //melody = 3 *  melody
    public static Melody multiplyInt(int time, Melody melody){

        Melody newMelody = new Melody(melody);
```

```
        for(int i=0;i<time;i++){
            newMelody = addMelody(newMelody,melody);
        }
        return newMelody;
    }

    // music = music + music
    public static Music addMusic(Music music_1, Music music_2)// music played
later
    {
        List<Track> trackList_2 = music_2.getTracks();
        List<Track> trackList_1 = music_1.getTracks();

        List<Track> newTrackList = new ArrayList<Track>();

        if((trackList_1 == null) || (trackList_2 == null))
                throw new NullPointerException();

        int maxEndTime = music_1.getTimeLength();


        newTrackList.addAll(trackList_1);

        for(Track track : trackList_2)
        {
            Melody melody = track.getMelody();
            if(melody == null)
                throw new NullPointerException();

            List<Note> newNoteList = new ArrayList<Note>();
            List<Note> noteLists = melody.getNoteList();
            if(noteLists == null)
                throw new NullPointerException();

            int startTime = maxEndTime;
```

```
            for(Note note: noteLists)
            {
                Note tmp = new Note(note);
                tmp.setStartTime(startTime);

                newNoteList.add(tmp);

                startTime = startTime + note.getDuration();
            }
            Melody newMelody = new Melody(newNoteList);
            Track newTrack = new Track(newMelody, track.getTimbre());
            newTrackList.add(newTrack);
            //
System.out.println(newTrack.getMelody().getNote(0).getStartTime()+"hhhh");
        }

        Music newMusic = new Music(newTrackList);

        return newMusic;

    }

    // music = music * music
    public static Music multiplyMusic(Music music_1, Music music_2)
    {
        List<Track> trackList_1 = music_1.getTracks();
        List<Track> trackList_2 = music_2.getTracks();

        List<Track> trackList = new ArrayList<Track>();

        if((trackList_1 == null) || (trackList_2 == null))
                throw new NullPointerException();

        for(Track track : trackList_2)
        {
```

```
            Melody melody = track.getMelody();
            if(melody == null)
                throw new NullPointerException();

            List<Note> noteLists = melody.getNoteList();
            if(noteLists == null)
                throw new NullPointerException();

        }
        trackList.addAll(trackList_1);
        trackList.addAll(trackList_2);

        Music newMusic = new Music(trackList);
        return newMusic;
    }

}
```

Main.java (for test)

```
package test;

import java.io.*;
import java.util.ArrayList;

import com.leff.midi.MidiFile;
import com.leff.midi.MidiTrack;
import com.leff.midi.event.MidiEvent;
import com.leff.midi.event.ProgramChange;
import com.leff.midi.event.meta.Tempo;
import com.leff.midi.event.meta.TimeSignature;
```

```
public class main {

    public static void main(String args[]){
        // 1. Create some MidiTracks
        MidiTrack tempoTrack = new MidiTrack();
        MidiTrack noteTrack = new MidiTrack();



        // 2. Add events to the tracks
        // 2a. Track 0 is typically the tempo map
        TimeSignature ts = new TimeSignature();
        ts.setTimeSignature(4, 4, TimeSignature.DEFAULT_METER,
TimeSignature.DEFAULT_DIVISION);

        Tempo t = new Tempo();
        t.setBpm(228);

        tempoTrack.insertEvent(ts);
        tempoTrack.insertEvent(t);

        int start = 0;
        int tempo = 1;
        int lasting = 200;
        // The 1st Channel, PIANO
        MidiEvent pc1 = new ProgramChange(0, 1, 0);
        noteTrack.insertEvent(pc1);
        for(int i = 0; i < 13; i++){
         noteTrack.insertNote(2, 60 + i, 120, start+tempo*lasting, tempo*lasting
+ 1600);
         start += tempo*lasting;
        }



        // It's best not to manually insert EndOfTrack events; MidiTrack will
        // call closeTrack() on itself before writing itself to a file
```

```
        // 3. Create a MidiFile with the tracks we created
        ArrayList<MidiTrack> tracks = new ArrayList<MidiTrack>();
        tracks.add(tempoTrack);
        tracks.add(noteTrack);

        MidiFile midi = new MidiFile(MidiFile.DEFAULT_RESOLUTION, tracks);

        // 4. Write the MIDI data to a file

        File output = new File("tmp.mid");
        try {
            midi.writeToFile(output);
        }
        catch(IOException e) {
            System.err.println(e);
        }
        System.err.println("haha");
    }



}
```

Test.java(for test)

```
package test;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
```

```
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;

import definition.*;
import function.PublicFunction;

import com.leff.midi.*;
import com.leff.midi.event.MidiEvent;
import com.leff.midi.event.NoteOff;
import com.leff.midi.event.NoteOn;
import com.leff.midi.event.ProgramChange;
import com.leff.midi.event.meta.Tempo;
import com.leff.midi.event.meta.TimeSignature;

public class test {

    public static Melody addChords1(Melody melody){
        Melody newMelody = new Melody();
        for(int i = 0;i < melody.getLength();
        i++)
        {
          Note note = melody.getNote(i);
        int pitch = note.getPitch() - 3;
        int duration = note.getDuration();
        int starttime = note.getStartTime();
        int strength = note.getStrength();
        Note chord = new Note(pitch, duration, starttime, strength);
        newMelody.addNote(chord);

        }
        return newMelody;
        }

        public static Melody addChords2(Melody melody){
```

```
        Melody newMelody = new Melody();
        int i = 0;
        while(true){
          Note note = melody.getNote(i);
        int pitch = note.getPitch() - 3;
        int duration = note.getDuration();
        int starttime = note.getStartTime();
        int strength = note.getStrength();
        Note chord = new Note(pitch, duration, starttime, strength);
        newMelody.addNote(chord);

        i++;

        if (i >= melody.getLength())break;
        }
        return newMelody;
        }

        public static void main(String arg[]){
            Music music = PublicFunction.read("twinkle_twinkle3.mid");
            Melody melody = music.getTrack(0).getMelody();
            Melody newMelody = addChords2(melody);
            List<Track> tracks = new
ArrayList<Track>(Arrays.asList(music.getTrack(0), new Track(newMelody,
C.PIANO)));
            Music newMusic = new Music(tracks);
            PublicFunction.write(newMusic, "new_twinkle_twinkle.mid");

            PublicFunction.print("Done...\n");
        }
}
```

testNote.java(for test)

```
package test;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import definition.*;
import function.*;

public class testNote {

    public testNote(int testcase){

        PublicFunction.setNoteDefault(C.C4, 0, 5, 50);
        List<Note> notes1 = new ArrayList<Note>(Arrays.asList(new Note(),new
Note(C.G4),
                new Note(C.A4),new Note(C.G4),new Note(C.F4),new Note(C.E4),
                new Note(C.D4),new Note()));

        Note note = new Note(C.F1);
        Note note_2 = new Note(C.C1);

        Melody melody = new Melody(new ArrayList<Note>(Arrays.asList(note)));
        Track track = new Track(melody);



        Music music = new Music(new ArrayList<Track>(Arrays.asList(track)));
        PublicFunction.write(music, "test_note_prev.mid");

        if(testcase == 1){
            //1) test plus :note = note + 1, raise pitch
            note = PublicFunction.plus(10,note);
            melody = new Melody(new ArrayList<Note>(Arrays.asList(note)));
            track = new Track(melody);
            music = new Music(new ArrayList<Track>(Arrays.asList(track)));
```

```
            PublicFunction.write(music, "test_plusNote.mid");
        }
        else if(testcase == 2){
            //2) test minus: note = note - 2, decrease pitch
            note = PublicFunction.minus(note,2);
            melody = new Melody(new ArrayList<Note>(Arrays.asList(note)));
            track = new Track(melody);
            music = new Music(new ArrayList<Track>(Arrays.asList(track)));

            PublicFunction.write(music, "test_minusNote.mid");
        }
        else if(testcase == 3){
            melody = PublicFunction.addNote(note, note_2);
            track = new Track(melody);
            music = new Music(new ArrayList<Track>(Arrays.asList(track)));

            PublicFunction.write(music, "test_addNoteNote.mid");
        }

    }

    public static void main(String[] args) {

        testNote test = new testNote(3);


    }

}
```

testMelody.java(for test)

```
package test;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import definition.*;
import function.*;


public class testMelody {

    private testMelody(int testcase)
    {

        PublicFunction.setNoteDefault(C.C4, 0, 250, 50);

        List<Note> notes1 = new ArrayList<Note>(Arrays.asList(new Note(),new
Note(C.G4),
                new Note(C.A4),new Note(C.G4),new Note(C.F4),new Note(C.E4),
                new Note(C.D4),new Note()));

        Melody melody = new Melody(notes1);
        Track track = new Track(melody,C.PIANO);
        Music music = new Music();
        music.insertTrack(track);
        PublicFunction.write(music, "test_melody_prev.mid");

        PublicFunction.setNoteDefault(C.C4, 0, 500, 50);
        List<Note> notes2 = new ArrayList<Note>(Arrays.asList(new Note(),new
Note(C.G4),
                new Note(C.A4),new Note(C.G4),new Note(C.F4),new Note()));
        Melody melody2 = new Melody(notes2);
        Melody newMelody = new Melody();
```

```
if(testcase == 1){
    //1) test addMelody in publicFunction
    newMelody = PublicFunction.addMelody(melody, melody2);

    track = new Track(newMelody,C.ACCORDION);
    music = new Music();
    music.insertTrack(track);
    PublicFunction.write(music, "test_addMelody_public.mid");
}
else if(testcase == 2){
    //2) test multiplyMelody in PublicFunction
    newMelody = PublicFunction.multiplyMelody(melody, melody2);
    track = new Track(newMelody,C.PIANO);
    music = new Music();
    music.insertTrack(track);
    PublicFunction.write(music, "test_multiplyMelody_public.mid");
}
else if(testcase == 3){
    //3) test addNote in PublicFunction
    newMelody = PublicFunction.addNote(newMelody, new Note(C.A7));
    track = new Track(newMelody,C.PIANO);
    music = new Music();
    music.insertTrack(track);
    PublicFunction.write(music, "test_addNote_public.mid");
}
else if(testcase == 4){
    //4) test multiplyInt in PublicFunction
    newMelody = PublicFunction.multiplyInt(melody2, 3);
    track = new Track(newMelody,C.PIANO);
    music = new Music();
    music.insertTrack(track);
    PublicFunction.write(music, "test_multiplyInt_public.mid");
}
else if(testcase == 5){
    //5) test subMelody
```

```
            newMelody = melody2.subMelody(0,melody2.getLength()-3);
            track = new Track(newMelody,C.PIANO);
            music = new Music();
            music.insertTrack(track);
            PublicFunction.write(music, "test_subMelody.mid");


        }
        else if(testcase == 6){
            //6) test deletNote
            melody.deleteNote(melody.getNoteList().size()-1);
            track = new Track(melody,C.PIANO);
            music = new Music();
            music.insertTrack(track);
            PublicFunction.write(music, "test_deleteNote.mid");
        }
        else if(testcase == 7){
            //7) test addMelody
            melody.addMelody(melody2);
            track = new Track(melody,C.ACCORDION);
            music = new Music();
            music.insertTrack(track);
            PublicFunction.write(music, "test_addMelody.mid");
        }
        else if(testcase == 8){
            //8) test multiplyInt
            melody.multiplyInt(3);
            track = new Track(melody,C.PIANO);
            music = new Music();
            music.insertTrack(track);
            PublicFunction.write(music, "test_multiplyInt.mid");
        }
        else if(testcase == 9){
            //9) test addMelody
            melody.multiplyMelody(melody2);
            track = new Track(melody,C.PIANO);
```

```
            music = new Music();
            music.insertTrack(track);
            PublicFunction.write(music, "test_multiplyMelody.mid");
        }
        else if(testcase == 10){
            //10) test addMelody
            newMelody.addNote(new Note(C.A7));
            track = new Track(newMelody,C.PIANO);
            music = new Music();
            music.insertTrack(track);
            PublicFunction.write(music, "test_addNote.mid");
        }
    }

    public static void main(String[] args){

        testMelody test = new testMelody(4);
    }
}
```

testMusic.java(for test)

```
package test;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import definition.C;
import definition.Melody;
import definition.Music;
import definition.Note;
import definition.Track;
import function.PublicFunction;
```

```
public class testMusic {

    public testMusic(int testcase){

        // 1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - ||
        PublicFunction.setNoteDefault(C.C4, 0, 250, 200);
        List<Note> notes0 = new ArrayList<Note>(Arrays.asList(new Note(), new
Note(),
                new Note(C.G4),new Note(C.G4),new Note(C.A4),new
Note(C.A4),new Note(C.G4,500),
                new Note(C.F4),new Note(C.F4),new Note(C.E4),new
Note(C.E4),new Note(C.D4),new Note(C.D4),
                new Note(C.C4,500)));

        // 1 - 5 - | 6 - 5 - | 4 - 3 - | 2 - 1 - ||
        PublicFunction.setNoteDefault(C.C4, 0, 500, 50);
        List<Note> notes1 = new ArrayList<Note>(Arrays.asList(new Note(),new
Note(C.G4),
                new Note(C.A4),new Note(C.G4),new Note(C.F4),new Note(C.E4),
                new Note(C.D4),new Note()));

        // Create melody
        Melody melody0 =  new Melody(notes0);
        Melody melody1 = new Melody(notes1);

        // Put into track list
        List<Track> tracks = new ArrayList<Track>(Arrays.asList(new
Track(melody0,C.PIANO),
                new Track(melody1, C.VIOLIN)));

        // Create music_1
        Music music_1 = new Music(tracks);

        // 1 1 5 5 | 6 6 5 - | 4 4 3 3 | 2 2 1 - ||
        PublicFunction.setNoteDefault(C.C4, 0, 250, 200);
```

```
        List<Note> notes2 = new ArrayList<Note>(Arrays.asList(new Note(), new
Note(),
                new Note(C.C4),new Note(C.A4),new Note(C.A4),new
Note(C.A4),new Note(C.E4,500),
                new Note(C.C4),new Note(C.A4),new Note(C.E4),new
Note(C.E4),new Note(C.D4),new Note(C.D4),
                new Note(C.C4,500)));

        // 1 - 5 - | 6 - 5 - | 4 - 3 - | 2 - 1 - ||
        PublicFunction.setNoteDefault(C.C4, 0, 500, 50);
        List<Note> notes3 = new ArrayList<Note>(Arrays.asList(new
Note(C.A4),new Note(C.G4),
                new Note(C.C4),new Note(C.F4),new Note(C.F4),new Note(C.A4),
                new Note(C.C4),new Note()));

        // Create melody
        Melody melody2 =  new Melody(notes2);
        Melody melody3 = new Melody(notes3);

        // Put into track list
        List<Track> tracks_2 = new ArrayList<Track>(Arrays.asList(new
Track(melody2,C.ACCORDION),
                new Track(melody3, C.FLUTE)));

        // Create music_2
        Music music_2 = new Music(tracks_2);


        if(testcase == 1){
            //test addMusic in PublicFunction
            Music newMusic = PublicFunction.addMusic(music_1, music_2);
            PublicFunction.write(newMusic, "test_addMusic_public.mid");
        }
        else if(testcase == 2){
            //test addMusic
```

```
            music_1.addMusic(music_2);
            PublicFunction.write(music_1, "test_addMusic.mid");
        }
        else if(testcase == 3){
            //test multiplyMusic n PublicFunction
            Music newMusic = PublicFunction.multiplyMusic(music_1, music_2);
            PublicFunction.write(newMusic, "test_multipleMusic_public.mid");
        }
        else if(testcase == 4){
            // test multiplyMusic
            music_1.multiplyMusic(music_2);
            PublicFunction.write(music_1, "test_multipleMusic.mid");
        }
    }

    public static void main(String[] args){

        testMusic test = new testMusic(4);
    }

}
```