

## Why MPL?

Break the traditional way to compose note by note Make the composition process programmable, efficient and interesting.

- Target Users
  - Professional musicians and music amateurs
  - People love both music and programming
- Sparking Features
  - Simple data structure
  - Convenient Operations
  - Rich Functions

# Time Plan

Task	ask Name					Feb			Mar										May					
		Dec	Jan 4	Jan 11	Jan 18	Jan 25	Feb 1	Feb 8	Feb 15	Feb 22	Mar 1	Mar 8	Mar 15	Mar 22	Mar	29 Ap	5 Ap	pr 12	Apr 19	Apr 26	May 3	May 10	May 17	May 24
1	1. Team up 2/4																							
2	1.1. Basic Idea and Development Environment																							
3	1.2 Member Responsibility																							
4	2. Prepare for whitepaper																							
5	2.1 Music Processor Development History Research																							
6	2.2 Tardet Users and New Features for MPL																							
7	2.3 Basic type and data structure																							
8	2.4 Basic Operations and Functions																							

1	1. Team up 2/4		
2	1.1. Basic Idea and Development Environment		
3	1.2 Member Responsibility		
4	2. Prepare for whitepaper		
5	2.1 Music Processor Development History Research		
6	2.2 Tardet Users and New Features for MPL		

2.5 Representative Programs

3.2 Full Gramar Design

4. Project Implementation 4.1 Ocaml Part 4.1.1 Basic study of Ocaml

4.1.4 AST implementation 4.1.5 Type Checking 4.1.6 Java Code Generation 4.2 Java Part 4.2.1 Google MIDI Study 4.2.2 Java Function Implementation

5 Test 5.1 Ocami Part Test 5.1.1 test for sanner and parser 5.1.2 test for scanner, parser and AST 5.1.3 test for scanner, parser, ast and typechecking

5.2 Java Part Test 5.3 Whole Test 6. Final Project Report 7. Demo

3. Prepare for Tutorial & Language Reference Manual 3.1 Lexical Elements and Syntax Analysis Design

3.3 Write Tutorial & Language Reference Manual

4.1.2 Lexical Elements Detailed Design and Implementation of Scanner 4.1.3 Syntax Analysis Detailed Design and Implementation of Parser

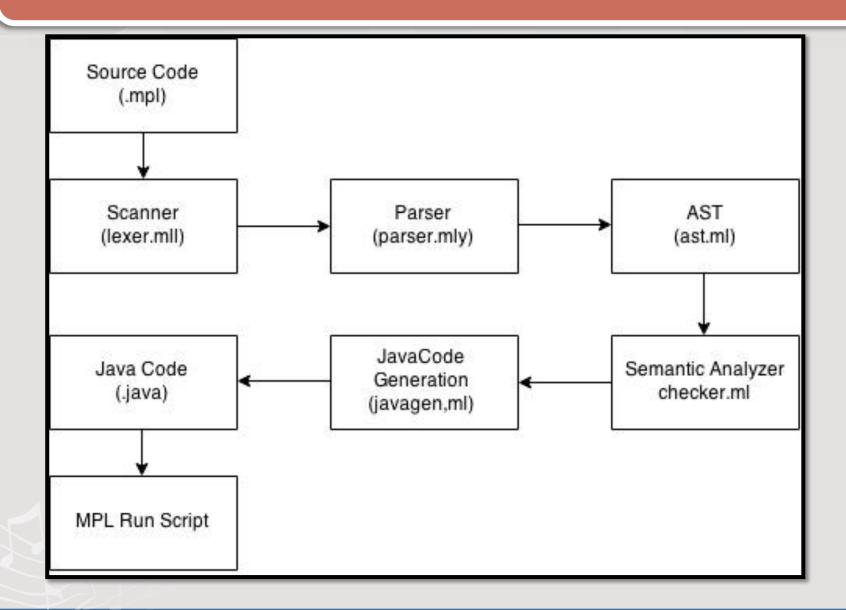
5.1.4 test for scanner, parser, ast, typechecking and code generation

### MPL in One Slide

```
Melody addChords(Melody melody) {
                                                                      Variable Definition Use
                 Melody newMelody = Melody();
Function
                                                                       Type Construction
                 for(int i = 0; i < melody.getLength(); i++) {</pre>
Definition
                     Note note = melody.getNote(i);
                                                                        Calculation of music
For Loop
                     Note chord = note - 12;
                     newMelody.addNote(chord);
                                                               Built-in function for music type
                 return newMelody;
             void main(string arg[]) {
                                                                          Built-in function for
                 Music music = read("twinkle twinkle0.mid");
                                                                          general purpose
Array
                 Melody melody = music.getTrack(0).getMelody();
Initialization
                 Melody newMelody = addChords(melody);
                 Track tracks[] = {music.getTrack(0), Track(newMelody, PIANO)};
                 Music newMusic = Music(tracks);
                                                                          Call user defined
Comments
                 write(newMusic, "new twinkle twinkle.mid");
                                                                          function
                 // Finished
```



### How it works



# Role & Responsibility

Module	Responsible Members
Scanner	Mengting Wu, Bo Wang
Parser	Yilin Xiong
AST	Bo Wang, Mengting Wu
Checker	Yilin Xiong, Ying Tan
Code Generation	Yilin Xiong, Bo Wang, Mengting Wu
Java Implementation	Shengyi Lin, Ying Tan
Test	Shengyi Lin

## Symbol Table

#### - Environment Stack

Keep Scope: Number each blocks.

Push the number of the current block when go into the block, and pop out after go out.

### - Symbol Table

Name each variable with name "scopeNumber\_name" and store its type with name into HashMap.

Type is stored as string.

Function type: void\_argType\_returnType

## Symbol Table

```
//Push: block 0 Stack: 0
void main(string arg[]) {
   // Push: block 1 Stack: 1 0
    Music music = read("twinkle_twinkle2.mid");
    int i;
    for(i = 0; i < music.getNumberOfTracks(); i++) {</pre>
        // Push: block 2 Stack: 2 1 0
        Track track = music.getTrack(i);
        track.setTimbre(PIANO);
        // Pop: block 2 Stack: 1 0
    write(music, "twinkle_twinkle3.mid");
    // Pop: block 1 Stack: 0
```

### Symbol Table:

Variable	Туре							
l_arg	array_string							
1_music	Music							
1_i	int							
2_track	Track							
main	void_array_s tring_void							

### Run Time Environment

- Ocaml (Frontend for translator)
- JDK 1.6 (JVM)
- MPL.jar: (Java backend with midi API)

users can download from https://github.com/PLT-MPL/MPL/tree/master/ShellScript to generate midi file.

#### MPL.sh

Help user to test MPL code more conveniently.

## Script and Test Plan

#### Command:

MPL.sh <Path/To/ShellScript/Folder> <Input/MPL/
Filename> <Output/Executable/Filename>

#### Test Plan:

Modules	Test Result
Hello World	$\checkmark$
Note Initialization	$\sqrt{}$
Note Manipulation	√
Melody Initialization	√
Melody Manipulation	$\checkmark$
Track Initialization	$\checkmark$
Music Initialization	$\checkmark$
Music Manipulation	$\checkmark$
Music Creation	
Self-defined Function	$\checkmark$



### Lessons Learnt

- Flexible timeline are vital important.
- Start work before considering too much.
- Functional languages are sooo interesting.
- We should have taken Computer Science
   Theory class before it.

