MPL

Music Processing Language

Project Manager: Bo Wang

Language Guru: Yilin Xiong

System Architect: Shengyi Lin

System Integrator: Ying Tan

Verification and Validation: Mengting Wu

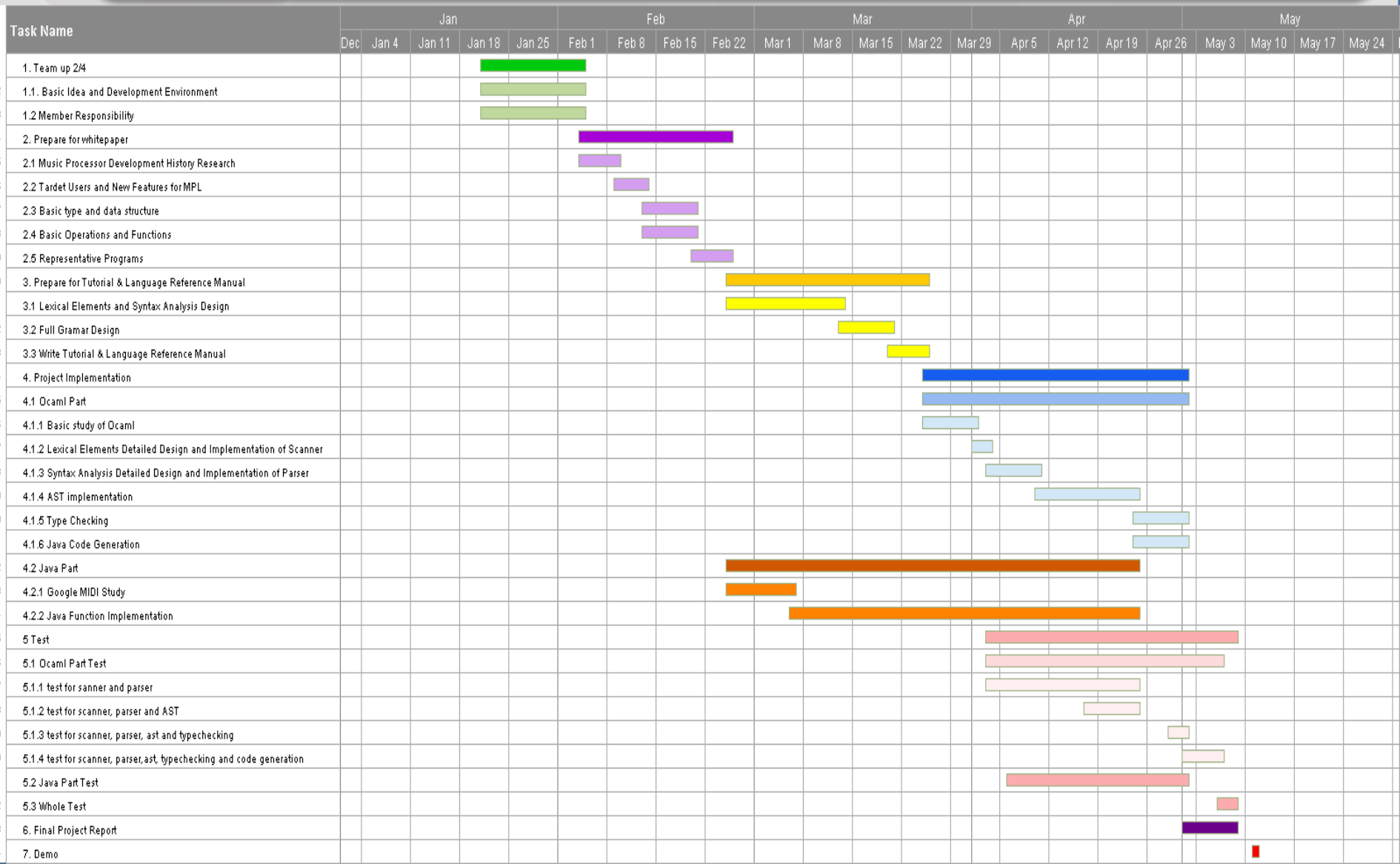
Why MPL?

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 Name
Jan
Feb
Mar
Apr
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
Apr 5
Apr 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
9 2.5 Representative Programs
10 3. Prepare for Tutorial & Language Reference Manual
11 3.1 Lexical Elements and Syntax Analysis Design
12 3.2 Full Gramar Design
13 3.3 Write Tutorial & Language Reference Manual
14 4. Project Implementation
15 4.1 Ocaml Part
16 4.1.1 Basic study of Ocaml
17 4.1.2 Lexical Elements Detailed Design and Implementation of Scanner
18 4.1.3 Syntax Analysis Detailed Design and Implementation of Parser
19 4.1.4 AST implementation
20 4.1.5 Type Checking
21 4.1.6 Java Code Generation
22 4.2 Java Part
23 4.2.1 Google MIDI Study
24 4.2.2 Java Function Implementation
25 5 Test
26 5.1 Ocaml Part Test
27 5.1.1 test for sanner and parser
28 5.1.2 test for scanner, parser and AST
29 5.1.3 test for scanner, parser, ast and typechecking
30 5.1.4 test for scanner, parser,ast, typechecking and code generation
31 5.2 Java Part Test
32 5.3 Whole Test
33 6. Final Project Report
34 7. Demo

MPL in One Slide

```
Melody addChords(Melody melody) {
    Melody newMelody = Melody();
    for(int i = 0; i < melody.getLength(); i++) {
        Note note = melody.getNote(i);
        Note chord = note - 12;
        newMelody.addNote(chord);
    }
    return newMelody;
}
void main(string arg[]) {
    Music music = read("twinkle_twinkle0.mid");
    Melody melody = music.getTrack(0).getMelody();
    Melody newMelody = addChords(melody);
    Track tracks[] = {music.getTrack(0), Track(newMelody, PIANO)};
    Music newMusic = Music(tracks);
    write(newMusic, "new_twinkle_twinkle.mid");
    // Finished
}
```

Function Definition

For Loop

Array Initialization

Comments

Variable Definition Use Type Construction

Calculation of music

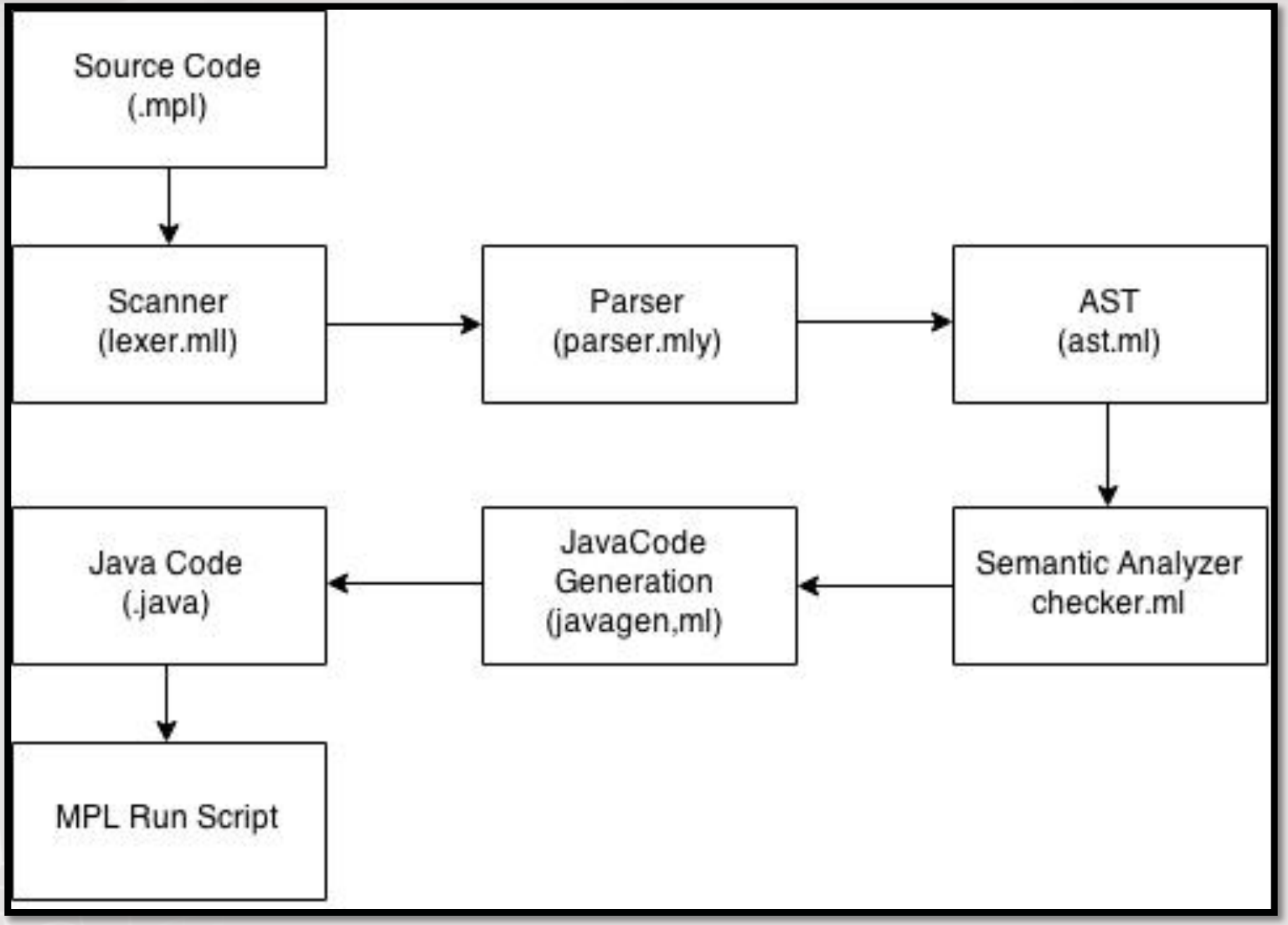
Built-in function for music type

Built-in function for general purpose

Call user defined function


Key Features Demo

How it works


Source Code
(.mpl)
Scanner
(lexer.mll)
Parser
(parser.mly)
AST
(ast.ml)
Semantic Analyzer
checker.ml
JavaCode
Generation
(javagen,ml)
Java Code
(.java)
MPL Run Script

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
// Push: block 1    Stack: 1 0
void main(string arg[]) {
    Music music = read("twinkle_twinkle2.mid");
    int i;
    // Push: block 2    Stack: 2 1 0
    for(i = 0; i < music.getNumberOfTracks(); i++) {
        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	Type
1_arg	array_string
1_music	Music
1_i	int
2_track	Track
main	void_array_string_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	√
Note Initialization	√
Note Manipulation	√
Melody Initialization	√
Melody Manipulation	√
Track Initialization	√
Music Initialization	√
Music Manipulation	√
Music Creation	√
Self-defined Function	√


DEMO

Lessons Learnt

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

Thanks!