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
```cpp
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
}
```
Key Features Demo
How it works

Source Code (.mpl) → Scanner (lexer.mll) → Parser (parser.mly) → AST (ast.ml) → Java Code (.java) → MPL Run Script

JavaCode Generation (javagen.ml) → Semantic Analyzer checker.ml
<table>
<thead>
<tr>
<th>Module</th>
<th>Responsible Members</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanner</td>
<td>Mengting Wu, Bo Wang</td>
</tr>
<tr>
<td>Parser</td>
<td>Yilin Xiong</td>
</tr>
<tr>
<td>AST</td>
<td>Bo Wang, Mengting Wu</td>
</tr>
<tr>
<td>Checker</td>
<td>Yilin Xiong, Ying Tan</td>
</tr>
<tr>
<td>Code Generation</td>
<td>Yilin Xiong, Bo Wang, Mengting Wu</td>
</tr>
<tr>
<td>Java Implementation</td>
<td>Shengyi Lin, Ying Tan</td>
</tr>
<tr>
<td>Test</td>
<td>Shengyi Lin</td>
</tr>
</tbody>
</table>
Symbol Table

- **Environment Stack**

  Keep Scope: Number each block.
  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
void main(string arg[]) {
    Music music = read("twinkle_twinkle2.mid");
    int i;
    for(i = 0; i < music.getNumberOfTracks(); i++) {
        Track track = music.getTrack(i);
        track.setTimbre(PIANO);
    }
    write(music, "twinkle_twinkle3.mid");
}
Run Time Environment

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


**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:

<table>
<thead>
<tr>
<th>Modules</th>
<th>Test Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World</td>
<td>✓</td>
</tr>
<tr>
<td>Note Initialization</td>
<td>✓</td>
</tr>
<tr>
<td>Note Manipulation</td>
<td>✓</td>
</tr>
<tr>
<td>Melody Initialization</td>
<td>✓</td>
</tr>
<tr>
<td>Melody Manipulation</td>
<td>✓</td>
</tr>
<tr>
<td>Track Initialization</td>
<td>✓</td>
</tr>
<tr>
<td>Music Initialization</td>
<td>✓</td>
</tr>
<tr>
<td>Music Manipulation</td>
<td>✓</td>
</tr>
<tr>
<td>Music Creation</td>
<td>✓</td>
</tr>
<tr>
<td>Self-defined Function</td>
<td>✓</td>
</tr>
</tbody>
</table>
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!