School of Computer Science, Sun Yat-sen University

MRL：一种单音轨 midi 音乐表示语言

19335070 黄靖雯

2022 年 10 月 13 日

目录

1 简介与背景

本语言是一种简单的基于 midi 的音乐表示语言，只需了解一些非常基本的乐理知识就可以很好地掌握本语言。经过编译，本语言的文件能够生成含有单个音轨的 midi 文件，任意 midi 播放器都能够将其内容准确地演奏出来。

尽管表达音乐的能力十分有限，本语言还是存在许多可能的应用场景。想象这样一个场景，你希望快速起草一段脑中的旋律，然而除了电脑以外你的手边没有任何乐器；这时，你就可以利用本语言来快速地编写生成 midi 文件，并立即聆听这段旋律在多种乐器上的模拟演奏效果。另一方面，本语言提供了一些全局的乐曲配置选项以及简单的控制结构，因此在给定乐曲本身的内容之后，本语言也有潜力为一些简单的风格化乐曲生成应用提供基础。比如根据来电人士的不同演奏不同风格的同一乐曲，从而使来电铃声本身更具提示性；再比如根据闹钟的推迟次数逐级增强铃声中的音乐情绪，从而更好地唤醒用户等等。

本课程设计的相关代码已开源：https://github.com/uangjw/Music-Representing-Language，各测试用例所对应的生成结果也可从此仓库中下载试听。

2 乐理知识

本语言所能表达的音乐较为简单，在此将其中涉及的基本乐理知识介绍如下。

2.1 乐音体系与音级

在音乐中使用的、有固定音高的音的总和，叫做乐音体系。乐音体系中的各音叫做音级，音级有基本音级和变化音级。

基本音级的名称可以用字母和唱名两种方式来标记，本语言使用字母体系来表示音级。

1. 字母体系：C D E F G A B

2. 唱名体系：1(do) 2(rei) 3(mi) 4(fa) 5(sol) 6(la) 7(si)

2.2 音高关系

钢琴上白键所发出的音是与基本音级相符合的。钢琴上 52 个白键循环重复地使用七个基本音级名称。两个相邻的具有同样名称的音叫做八度，通过在基本音级的名称后附加音阶编号，可以指定该音级具体是哪个八度上的（如 C3 表示第 3 个八度上的 C）。

升高或降低基本音级而得来的音，叫做变化音级。半音是音高关系中最小的计量单位，两个半音即为一个全音；钢琴上两个相邻白键之间的音高间隔为一个全音，相邻白键与黑键之间的音高间隔为一个半音。将基本音级升高半音在五线谱中用“#”来标明；降低半音用“b”来标明。

2.3 节拍与乐曲速度

节拍是衡量节奏的单位。在音乐中，有一定强弱分别的一系列拍子在每隔一定时间重复出现；节拍就是在乐曲中表示固定单位时值以及这种强弱规律的组织形式。

表示每小节中基本单位拍的时值和数量的记号，称为拍号。拍号上方的数字表示每小节的拍数，下方数字表示每拍的时值。例如，2/4 表示已 4 分音符为一拍，每小节两拍。拍号中时值的实际时间，视乐曲所标速度而定。

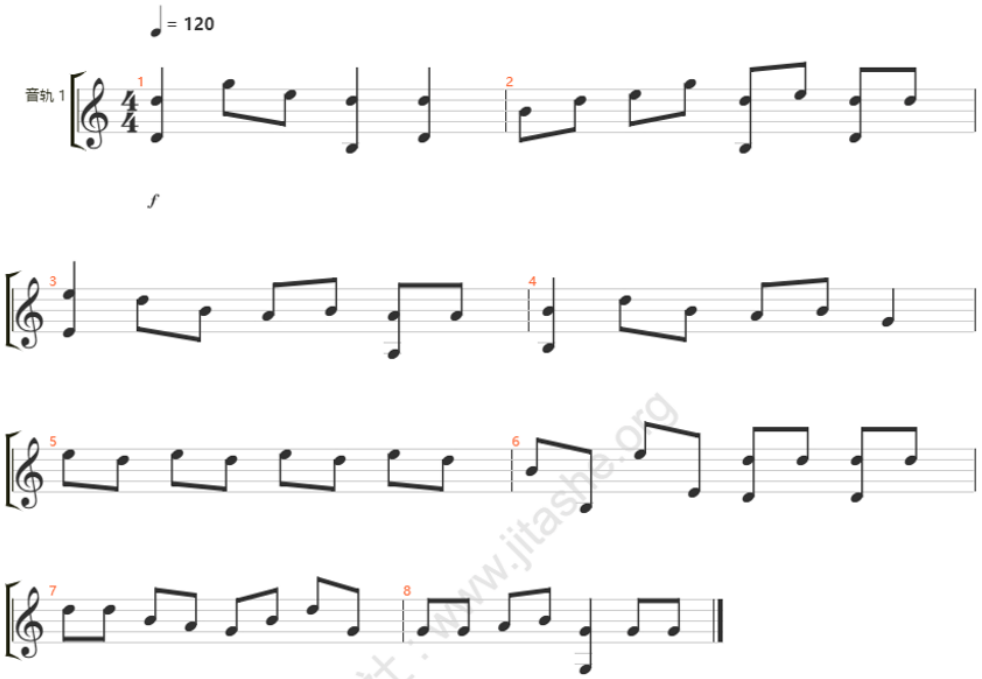
BPM（Beat Per Minute）即每分钟节拍数，是全乐曲的速度标记，是独立在曲谱之外的速度标准。

拍、BPM 以及拍号的关系：BPM 描述的是时钟速度，对应的是具体时间下应有多少拍。拍是固化的曲谱记号，通常认为一个四分音符即是一拍，其余所有音符严格参照一拍进行分割。因此，拍实则是关联曲谱音符与时间的纽带，本身不具有时间与速度的属性。拍号是曲谱书写规范的标记，其描述的是一个小节可以谱写多少拍的音符，而不是时间或速度。

3 语言基本设计

现以儿歌《小兔子乖乖》为例介绍本语言的基本设计。

下图为吉他演奏的《小兔子乖乖》的五线谱。首先可以在乐谱中看到乐曲的全局信息：BPM 为 120，拍号为 4/4，全曲音高在中央 C 附近（即全曲的音高都在音阶 3 附近），一共有 8 个小节。如拍号所规定的，乐曲的每一个小节中都只有 4 个四分音符（一个四分音符可能分割成两个八分音符）。



下面给出《小兔子乖乖》的 MRL 表示。可以看到，程序首先定义了乐曲的全局信息，包括全曲速度（BPM），节拍（TEMPO），使用乐器（INSTRUMENT），演奏强度（VELOCITY）以及当前音高所在的音阶（CUR_OCTAVE）。接着是乐曲的主体，一个音乐段（MUSIC），用“bunny”作为它的标识符。在音乐段中，以“=”标识的语句都是对乐曲各个小节内容的描述。由于节拍设置为 4，所以每一个小节中都有 4 个四分音符的拍值，拍与拍之间用“|”分隔。在一个“拍”的区域中，存在着可能由括号括起的字符序列，这些就是具体发声的音符；一个音符由音名（大写字母）与后缀共同组成，括号括起的多个音符同时发声，构成一个和弦。

```
BPM: 120;
INSTRUMENT: 25;        //Acoustic Guitar(steel)
CUR_OCTAVE: 3;
VELOCITY: 1;
TEMPO: 4;

MUSIC bunny {
    = (DD+) | G+E+ | (B-D+) | (DD+) ;
    = BD+ | E+G+ | (D+B-)E | (DD+)D+ ;
```

```
        = (EE+) | D+B | AB | (A–A)A ;
        = (B–B) | D+B | AB | G ;
        = E+D+ | E+D+ | E+D+ | E+D+ ;
        = BB– | E+E | (DD+)D+ | (DD+)D+ ;
        = D+D+ | BA | GB | D+G ;
        = GG | AB | (G–G) |GG ;
    }
```

4 语言定义

4.1 词法定义

4.1.1 正则定义式

现将本语言的词法定义以正则定义式的形式总结如下：

$$note\_name \rightarrow C|D|E|F|G|A|B$$
$$letter \rightarrow A|B|...|Z|a|b|...|z$$
$$lower\_case\_letter \rightarrow a|b|...|z$$
$$digit \rightarrow 0|1|2|3|4|5|6|7|8|9$$
$$identifier \rightarrow lower\_case\_letter(letter|digit)*$$

4.1.2 标识符

标识符（identifier）是一个小写字母开头的，由数字与小写大写字母组成的字符序列，其长度不超过 24 个字符。本语言的标识符是大小写敏感的。

标识符可以用于表达如下实体：

1. 整数对象
2. 段落段（segment）
3. 音乐段（music）；注，音乐段的标识符也将是编译生成的 midi 文件的文件名

其中，整数对象以及段落段的标识符都是可以在后续进行多次引用的。一个例子展示如下（省略了乐曲全局配置）：

```
SEGMENT seg1{
    = C | E | G ;
}

SEGMENT seg2{
    = D | F | A ;
}

MUSIC song{
    int i: 3;
    if (i > 0) {
        seg1; // 将只演奏seg1的内容
    }
    else {
```

```
            seg2;
        }
    }
```

4.1.3 类型

本语言较为简单，只有三种类型的实体：

1. 音符（note）
2. 和弦（chord）
3. 整数（integer）

音符类型

音符类型的实体由一个音名（name）与后缀（postfix）构成。

音名即乐理中的 7 个音名，只能在此 7 个大写字母中取其一表示音名：C，D，E，F，G，A，B。

后缀包括：

1. +：在当前八度中音名相应音高的基础上再升一个八度
2. -：在当前八度中音名相应音高的基础上再降一个八度
3. #：在音名相应音高的基础上升一个半音

其中，区别于五线谱，‘#’的作用不能叠加，后缀中‘#’的个数为单数时上升一个半音，为双数时没有变化。

一个例子展示如下：

```
BPM: 120;
INSTRUMENT: 25;       //Acoustic guitar(steel)
CUR_OCTAVE: 3;
VELOCITY: 1;
TEMPO: 1;

MUSIC noteExample{
    = C;      // 即C3
    = C+;     // 即C4
    = C-;     // 即C2
    = C#;     // 即升C3
    = A+#;    // 即升A4；n5到n7表明各后缀表示符号
              // 可以叠加，且没有顺序之分
    = A+-;    // 即A3
```

```
    = E#++; // 即升E5
    = G##;  // 即G3
}
```

为了形成丰富的节奏，本语言还定义了两种特殊的音符“.”与“%”：

1. “.”：表示延续上一发声内容
2. “%”：休止符，即不发声

一个例子展示如下：

```
BPM: 120;
INSTRUMENT: 25;      //Acoustic guitar(steel)
CUR_OCTAVE: 3;
VELOCITY: 1;
TEMPO: 4;

MUSIC specialNoteExample{
    = C | . | . | . ;    //即一个延续4拍的C3
    = % | % | % | % ;    //即连续4拍不发声
    = C% | D% | E% | F% ;    //即演奏半拍C3后停半拍，
                             //再演奏半拍D3后停半拍，
                             //再演奏半拍E3后停半拍，
                             //再演奏半拍F3后停半拍
    = C | .D | .E | .F ;     //即演奏一拍半C3后演奏一拍D3，
                             //再演奏一拍E3，再演奏半拍F3
}
```

和弦类型

和弦类型实体表示多个音符在同一时刻发声，并持续相同的一段时间；和弦类型实体通过在括号中填入多个音符来构造。

一个例子展示如下：

```
BPM: 120;
INSTRUMENT: 25;      //Acoustic guitar(steel)
CUR_OCTAVE: 3;
VELOCITY: 1;
TEMPO: 1;

MUSIC chordExample{
    = (C)(E)(G);     // 即C3、E3与G3依次发声
```

```
    = (CEG);                    // 即C3、E3与G3同时发声
}
```

整数类型

整数类型可以显式地声明，并进行赋值操作；本语言实现了简单的整数四则运算。

一个例子展示如下（若运行，将生成没有声音内容的 midi 文件）：

```
BPM: 120;
INSTRUMENT: 25;       //Acoustic guitar(steel)
CUR_OCTAVE: 3;
VELOCITY: 1;
TEMPO: 1;

MUSIC intExample{
    int i : 0;        //声明一个整数变量i；本语言的声明必须同时初始化
    int j : i + 1;    //j值为1
    int k : j * 3;    //k值为3
    int z : k / 2;    //z值为1（1.5化为整数）
}
```

4.2 语法定义

本语言完整的 EBNF 定义请见附录一。

4.3 语言描述

本节提供描述本语言时指定用词的定义和所用概念。

4.3.1 注释

本语言允许行尾注释，编译器在编译时将略过它们。

```
// comment
```

4.3.2 标点符号

本语言中含有的标点符号及它们的作用如下：

标点符号	作用
{ }	1）为段结构划定界限 2）为控制结构划定界限
( )	1）全局设置的 SET 函数以及构造和弦对象的函数调用操作符 2）在控制语句中，为控制表达式划定界限
;	1）表明一个小节的结束 2）表明一个声明的结束
=	1）表明一个小节的开始
:	1）赋值运算符 2）在全局设置中标识设置内容的开始
//	1）表明一个行尾注释的开始
+	1）加法运算符 2）标识音符为当前八度中对应的音符再升八度
-	1）减法运算符 2）标识音符为当前八度中对应的音符再降八度
	1）乘法运算符
/	1）除法运算符
==	1）相等逻辑运算符
!=	1）不等逻辑运算符
>	1）大于逻辑运算符
<	1）小于逻辑运算符
>=	1）大于等于逻辑运算符
<=	1）小于等于逻辑运算符
#	1）标识一个音符为音名对应音符再升一个半音
|	1）在一个小节中分隔相邻两个拍子
.	1）标识延音节拍，即延续相邻节拍中的音符
%	1）标识休止符，即表示这一拍上没有发声的音符

4.3.3 音乐段

本语言的每一个程序都有且仅有一个音乐段（music），类似于 C 语言的 main 函数，是必须含有的内容，也是生成音乐的开始之处。音乐段以 MUSIC 作为标识，其内容本质上是一系列相互穿插的小节（section）与语句（statement）；小节是具体的演奏内容，语句则包括声明、赋值、段（segment）的调用以及选择控制（if-else）等。音乐段的标识符用于作为编译生成的 midi 文件的文件名。

```
MUSIC song{
    // body
}
```

4.3.4 段落段

本语言的每一个程序都可以有零到多个段落段（segment），类似于 C 语言的函数。段落段以 SEGMENT 作为标识，其内容只能是一系列的小节，不允许含有语句。音乐段可以通过段落段的标识符来进行调用。段落段之间不可以相互调用，也不可以嵌套定义。

```
SEGMENT seg{
    // body
}

MUSIC song{
    seg; // 调用seg段中的内容
}
```

4.3.5 关键字

本语言预留了如下三种以小写字母开头的关键字。由于语言已经使用了这些关键字，所以不允许在程序中使用与它们同名的标识符。

关键字	用途
int	整数类型
if	条件语句
else	条件语句

4.3.6 全局配置项

本语言的程序的文本文件开头处需完成对乐曲全局属性的配置。待配置的属性包括乐曲速度（BPM）、使用乐器音色（INSTRUMENT）、当前音阶（CUR_OCTAVE）、演奏强度（VELOCITY）以及节奏（TEMPO）。这些设置在源文件中都是必须的，且语句顺序也是固定的；如果全局配置语句有缺漏或者顺序不对，文件编译时都会报错。

乐曲速度

乐曲速度通过如下语句进行配置：

```
BPM: 120;
```

BPM 即每分钟所演奏的节拍数，为正整数。从直观感受来说，BPM 越小乐曲演奏得越慢，BPM 越大乐曲演奏得越快。

乐器

演奏乐曲时使用的乐器音色通过如下语句进行配置：

```
INSTRUMENT: 25;
```

INSTRUMENT 的配置值是 0 到 128 的整数，对应 General MIDI（GM）标准的旋律音色表序号。一些常用音色及其 GM 音色序号列表如下：

序号	音色
0	大钢琴（声学钢琴）
2	电钢琴
10	八音盒
13	木琴
21	手风琴
22	口琴
24	声学吉他（尼龙弦）
25	声学吉他（钢弦）
40	小提琴
56	小号
64	高音萨克斯
73	长笛

音阶

演奏乐曲时各音所在音阶（所在八度）通过如下语句配置：

```
CUR_OCTAVE:  3;
```

CUR_OCTAVE 的配置值为 0 到 8 的整数。实际上 midi 标准通过一个音符对应表来表示各音符，大致形式如下：

序号	音阶	音符
0	-1	C
1	-1	C#
2	-1	D
3	-1	D#
4	-1	E
5	-1	F
6	-1	F#
7	-1	G
8	-1	G#
9	-1	A
10	-1	A#
11	-1	B
12	0	C

midi 音符表中的音阶范围为-1 到 9。需要指出的是，midi 音符表中给出的音阶比一般乐理描述的音阶（钢琴上的八度）要高一级，例如一般乐理所指的“中央 C”为 C3，而 midi 音符表中 60 号音符的音高才是“中央 C”，在表中属于音阶 4。CUR_OCTAVE 的设置遵从一般乐理。

由于本版本没有做负数的实现，所以最小的音阶设置值为 0。

强度

演奏乐曲的强度通过如下语句配置：

```
VELOCITY: 1;
```

VELOCITY 的配置值只有 0 和 1,0 表示弱演奏力度,1 表示强演奏力度。推荐总是将 VELOCITY 设置为 1。

节奏

乐曲的节奏规定通过如下语句配置：

```
TEMPO: 4;
```

TEMPO 的配置值对应于接下来的各小节中的节拍数。TEMPO 为 4 时，乐曲中所有的小节中都必须含有 4 个拍子，而 TEMPO 为 3 时乐曲中所有小节都必须含有 3 个拍子。TEMPO 的设置是自由的，从本质上来说，如何划分小节并不能影响乐曲的节奏；统一地为全曲划分小节是为了辅助人们表达乐曲节奏。我们当然也可以将一首三拍子的圆舞曲按照 TEMPO 设置为 4 的格式来编写，只不过这样会使得乐曲在音乐角度上的可读性变差。

4.3.7 小节与语句

本语言中的语句可以大致分为顺序执行的表示小节的语句和其他语句两种。

小节

小节描述了一个音乐演奏的内容片段，是本语言源程序中最重要的一种语句。以 TEMPO 设置为 4 的乐曲为例，乐曲的一个小节中描述了四拍时间内的演奏内容，拍与拍之间以“|”划分，并以“=”标记一个小节的开头，如下面例子所示：

```
TEMPO: 4;

MUSIC sectionExample {
    = C | D | E | F; // 一个小节
}
```

语句

本语言中，非小节的语句包括整数的声明语句与赋值语句，if-else 语句，段落段（segment）的引用语句以及特殊的 SET_OCTAVE 语句。

整数的声明与赋值语句的例子可见于前文，在此不作赘述。

if-else 语句的例子如下。语句的 if 部分括号中必须为一个布尔表达式，else 部分不可以独立于 if 部分存在。if-else 语句中可以出现小节、整数声明与赋值、段落引用以及 SET_OCTAVE 语句；如果出现 if-else 的嵌套，语法分析器不会报错，但生成的音乐不能够满足嵌套的 if-else 语句的期望，这是一个待处理的实现漏洞。

```
if (i == 0) {
```

```
        = D | F | A ;
        i = 1;  // if-else 语句中允许所有 if-else 类型以外的语句存在
    }
    else {
        = E | G | C ;
    }
```

段落段的引用语句即段落段的标识符以及行尾“;”组成的语句，例子如下：

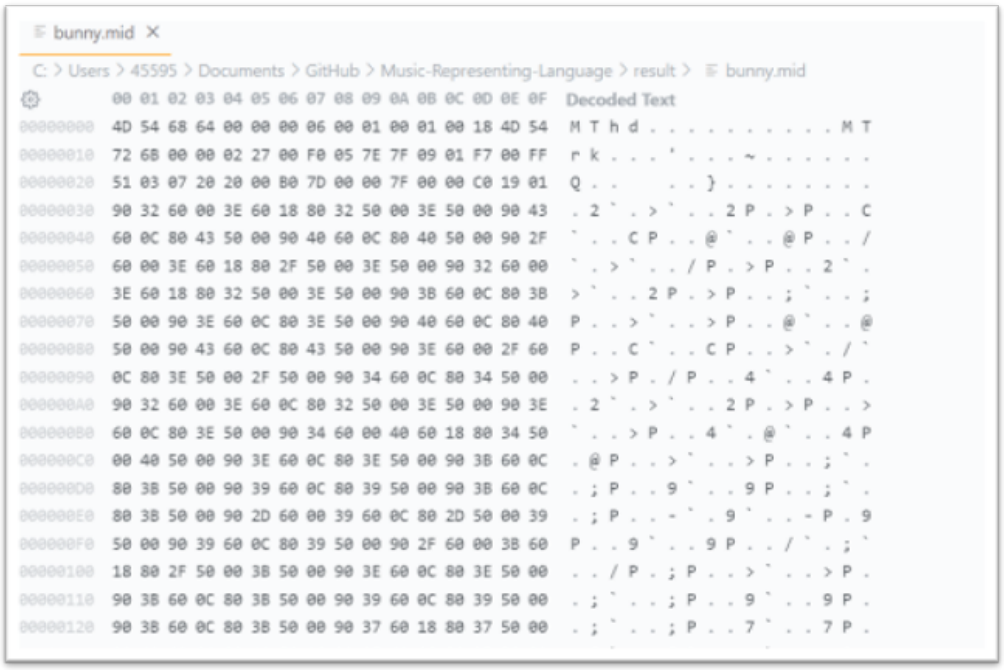
```
    seg1;
```

SET_OCTAVE 语句将修改乐曲的全局音阶设置,可以将其看作是对预定义函数 SET_OCTAVE 的调用，函数参数为一个整数，即新的音阶设置值：

```
    = C | C | C;    // 假设目前音阶设置为3，则此处演奏三拍C3
    SET_OCTAVE(2); // 将乐曲音阶设置为2
    = C | C | C;    // 演奏三拍C2
```

5 语言实现

本语言编译生成的结果即 midi 文件，midi 文件可以被 midi 播放器解析并播放。前文所提的例子《小兔子乖乖》所编译生成的 midi 文件部分内容如下图所示：



5.1 词法分析

本语言的词法分析器是利用 JFlex 工具自动产生的。JFLex 是一个类似 Unix 平台上 lex 程序的开源软件工具，其本身采用 Java 语言编写，并且生成 Java 语言的词法分析程序源代码。JFlex 的输入源文件根据本语言的词法规则编写。

本实现的词法分析器主要完成以下工作：

1. 识别空白符，不返回符号
2. 识别行尾注释，不返回符号
3. 识别各标点符号，返回各标点对应的符号
4. 识别“MUSIC”“int”等保留字，返回各保留字对应的符号
5. 识别整数（大于 12 位时报错），返回整数对应的符号以及该整数的值
6. 识别“C”“E”等音名，返回音名对应的符号以及该音名的值
7. 识别标识符（长于 24 个字符时报错），返回标识符对应的符号以及该标识符的值

成功识别时，词法分析器以字符串形式所返回的符号具体值；当词法分析器发现识别错误时，整个编译生成过程将直接终止。本词法分析器的异常处理主要依赖于 JFlex 内置的异常处理机制，即直接对字符匹配进行报错，尚未实现更细致的异常处理。

5.2 语法分析与翻译模式

本语言的语法分析器是利用 JavaCUP 工具自动产生的。JacaCUP 是一个开源的语法分析程序自动生成工具，属于 LALR Parser Generator，类似 Unix 平台上的 yacc 程序。JavaCUP 本身采用 Java 编写，并且生成 Java 语言的分析程序源代码。

本实现的语法分析器能够完成以下工作：

1. 出现语法错误时，主要依赖 JavaCUP 内置的异常处理机制进行报错；出现语义错误时，将抛出更具体的异常（节奏匹配异常、音符构造异常等）

2. 对于一个词法、语法和语义完全正确的源程序，自动生成一个 midi 文件；需要指出的是，语法分析器只从本语言的语法和语义层面发现错误，一些不符合 midi 标准的错误（如音阶取值超出范围）将不会被发现，并直接体现在生成的 midi 文件的播放效果中

由于 JavaCUP 是一种 LALR Parser Generator，因此本语言的翻译模式是一种适用于自底向上分析的后缀翻译模式。

5.3 语义动作实现

写 midi 文件的语义动作是通过语法分析器中一个 MidiWriter 成员对象实现的。MidiWriter 及其他语义动作相关的类与方法封装于程序包 write_midi 中，其中实现了类 MidiWriter 以及类 Note，相关方法的简要描述如下：

```
程序包 write_midi
    类
    MidiWriter  // 利用javax.sound.midi库来写midi文件的类
        方法
        addChord    // 向音轨中添加一个和弦
        addChord    // 重载，向音轨中特定位置添加一个和弦
        addNote     // 向音轨中添加一个音
        addNote     // 重载，向音轨中特定位置添加一个音
        addRest     // 向音轨中添加一个休止音
        addSequence// 向音轨中添加一段演奏（段落调用）
        getLastTick// 返回音轨最后一个音/和弦事件的开始时刻
        getNoteID   // 查midi音符表，返回特定音的序号
        writeMidiFile // 将音轨写入midi文件
    类
    Note          // 统一表示单音、休止符等四类音符
```

```
方法
getName     // 返回音符的音名
getOctave   // 返回音符所在音阶
isContinue  // 返回音符是否为延音标志
isNote      // 返回音符是否为单音
isNull      // 返回音符是否为无意义占位标志
isRest      // 返回音符是否为休止符
isSharp     // 返回音符是否升音
printInfo   // 打印Note对象信息（debug）
```

6 测试与分析

下面展示本语言的处理程序对某些源程序的编译生成输出。由于本语言编译生成的结果为 midi 文件,不便展示于文档中,因此建议读者从 https://github.com/uangjw/Music-Representing-Language 直接下载试听附录二中各用例编译生成的结果。

6.1 《小兔子乖乖》

《小兔子乖乖》的本语言源程序可见于附录二。由于没有词法、语法以及语义错误，编译生成时将在终端输出语法分析过程中各非终结符的归约结果，如下所示（完整输出较长，以下仅展示部分输出）：

```
...
reducing to sec_part
reducing to note_postfix
reducing to note_postfixes
reducing to note
reducing to chord_part
reducing to note
reducing to chord_part
reducing to chord
reducing to sound
reducing to beat
reducing to sec_part
reducing to note
reducing to sound
reducing to beat
reducing to note
reducing to sound
reducing to beat
reducing to sec_end
reducing to section
reducing to music_part
reducing to music
reducing to mrl_file
finish
```

以上输出均可与附录一中 EBNF 形成对应。归约到开始符号“mrl_file”后，语法分析器调用 MidiWriter 的 writeMidiFile 方法完成 midi 文件的生成。

6.2 含有词法错误的源程序及其输出

简单修改《小兔子乖乖》的源程序，将全局配置中预留字“INSTRUMENT”错误拼写，运行编译生成程序得到如下输出：

```
reducing to bpm_config
Lexical error matching I
Exception caught: Mrl lexical error at 1, 0
Stack trace:
Mrl lexical error at 1, 0
        at Scanner.next_token(Scanner.java:1077)
        at Parser.scan(Parser.java:324)
        at java_cup.runtime.lr_parser.parse(lr_parser.java:693)
        at Mrl.main(Mrl.java:10)
finish
```

可见 BPM 的设置成功归约，终端有输出“reducing to bpm_config”；词法异常由词法分析器 Scanner 抛出，形式为 JFlex 工具所定义，程序直接终止。

6.3 含有语法错误的源程序及其输出

简单修改《小兔子乖乖》的源程序，将唯一的音乐段改为段落段，即将“MUSIC”改为“SEGMENT”，运行编译生成程序得到如下输出（完整输出较长，在此只展示部分）：

```
...
reducing to note
reducing to sound
reducing to beat
reducing to note
reducing to sound
reducing to beat
reducing to sec_end
reducing to section
Syntax error at character 15 of input
instead expected token classes are []
Couldn't repair and continue parse at character 15 of input
Exception caught: java.lang.Exception:
Can't recover from previous error(s)
Stack trace:
java.lang.Exception: Can't recover from previous error(s)
        at java_cup.runtime.lr_parser
```

```
        .report_fatal_error(lr_parser.java:392)
        at java_cup.runtime.lr_parser
        .unrecovered_syntax_error(lr_parser.java:539)
        at java_cup.runtime.lr_parser
        .parse(lr_parser.java:731)
        at Mrl.main(Mrl.java:10)
finish
```

可见完成各小节的归约后，语法分析器抛出了异常，其形式为 JavaCUP 所定义，并不能明确地指出具体的语法错误内容。

6.4 含有语义错误的源程序及其输出

简单修改《小兔子乖乖》的源程序，加入一个休止符，并在其后试图用一个延音符号来对其进行延续，运行编译生成程序得到如下输出：

```
reducing to bpm_config
reducing to ins_config
reducing to curoc_config
reducing to veloc_config
reducing to tempo_config
reducing to config_statement
reducing to mrl_file_part
reducing to music_part
reducing to sound
reducing to beat
reducing to sec_part
reducing to sound
reducing to beat
reducing to sec_part
Exception caught: Cannot use continue symbol '.' after rest symbol '%'.
Stack trace:
Cannot use continue symbol '.' after rest symbol '%'.
        at Parser$CUP$Parser$actions
        .CUP$Parser$do_action_part00000000(Parser.java:1170)
        at Parser$CUP$Parser$actions
        .CUP$Parser$do_action(Parser.java:1677)
        at Parser.do_action(Parser.java:285)
        at java_cup.runtime.lr_parser.parse(lr_parser.java:699)
        at Mrl.main(Mrl.java:10)
```

```
finish
```

可以从调用栈中看到，异常是从语法分析器的 actions 抛出的，即此语义错误是具体地在翻译模式中定义的，因此异常描述能够准确地给出“不能在休止符‘%’之后使用延音符号‘.’”。

从以上三个简单的测试可以看出，本语言目前的实现所具有的异常处理机制还很粗糙，尤其不能给出错误的具体描述，因此还需要考虑进一步的完善。

7 总结与致谢

本语言是一种基于 midi 的单音轨音乐表示语言，语言设计相对简单，所涉及的乐理也比较简单。语言的核心要素包括全局设置，音乐段 music 与段落段 segment，小节、音符与和弦，以及一些简单的控制结构。

在最开始的阶段，我对语言设计方面的知识缺乏概念，也比较盲目地试图去参考一些已有的工作。后来，通过打开格局，放飞想象力，我渐渐在书写用例的过程中形成了一个基本的设计思路，并在老师的帮助和指导下不断地对语言设计进行完善和精简，才最终形成了如今的结果。在实现阶段，网络上与 midi 标准的解析相关的材料非常有限，因此我实际上是通过实验一点点了解 midi 文件各个字节的含义的；总的来说我对 midi 标准的了解并不系统，这给目前的实现结果留下了不少的改进空间。

完整的课程设计做下来，我感到自己对“语言是一种解决问题的方法”这一命题有了更深的感受。在此我也必须好好感谢过程中鼓励过我的同学们，以及一直非常耐心地指导我，并且从不吝啬夸奖的李老师。在最开始的语言设计阶段，老师曾对我说过一句话：“要做有趣的事情”，这深深地触动了我；我期望自己能够保持在这一课程设计中表现出的热情和创造力，继续去做更多的有趣的事情。

8 附录一：MRL 语言的 EBNF 定义

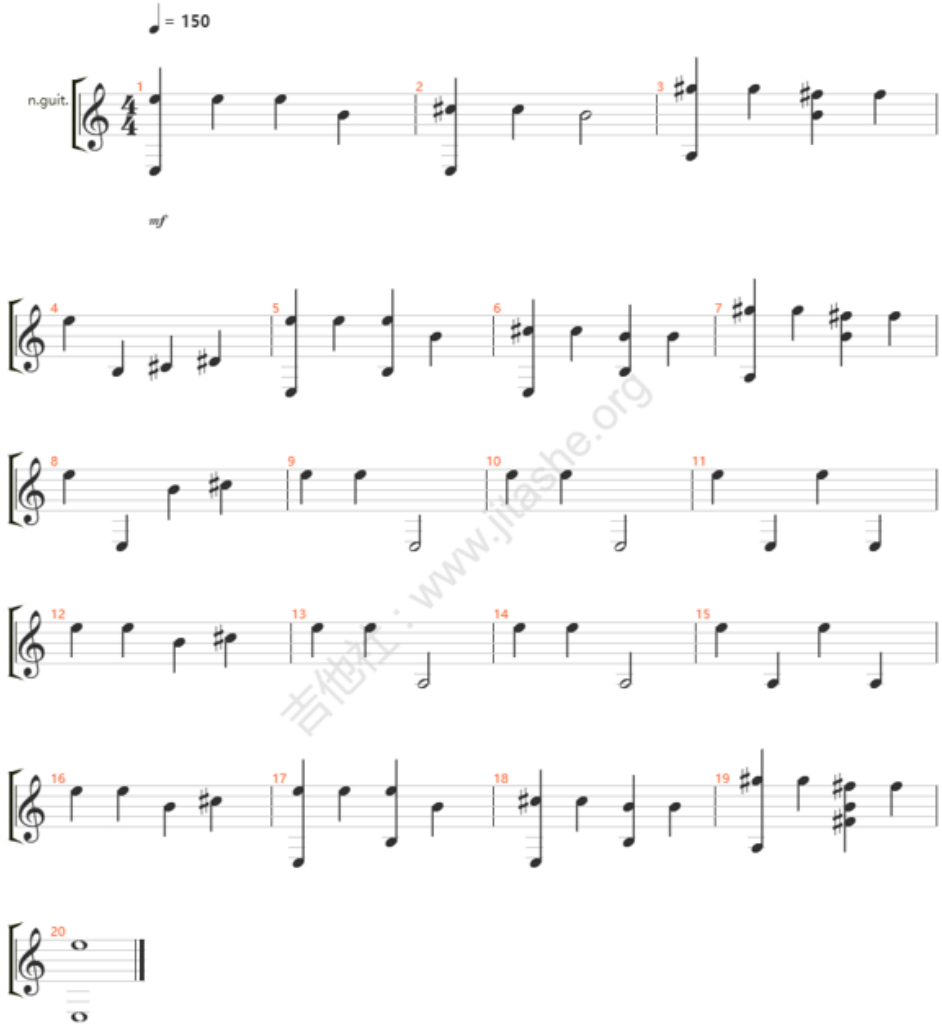
```
comment             = '//' {input_char} ;

mrl_file            = config_statement {segment} music ;

config_statement    = bpm_config ins_config curoc_config veloc_config
                      tempo_config ;

bpm_config          = 'BPM' ':' integer ';' ;

ins_config          = 'INSTRUMENT' ':' integer ';' ;

curoc_config        = 'CUR_OCTAVE' ':' integer ';' ;

veloc_config        = 'VELOCITY' ':' integer ';' ;

tempo_config        = 'TEMPO' ':' integer ';' ;

segment             = 'SEGMENT' identifier '{' {section} '}' ;

music               = 'MUSIC' identifier '{' {statement | section}
                      '}' ;

statement           = int_declaration | int_assignment | if_statement
                      | set_octave | seg_call ;

seg_call            = identifier ';' ;

set_octave          = 'SET_OCTAVE' '(' integer ')' ';' ;

int_declaration     = 'int' identifier ':' int_expr ';' ;

int_assignment      = identifier ':' int_expr ';' ;

if_statement        = 'if' '(' bool_expr ')' '{' {section |
                      statement} '}' ['else' '{' {section |
                      statement} '}'] ;
```

```
int_expr            = int_expr ('+' | '-' | '*' | '/') int_expr |
                      '(' int_expr ')' | identifier | integer ;

bool_expr           = int_expr ('==' | '>' | '<' | '>=' | '<=' |
                      '!=') int_expr ;

section             = '=' beat {'|' beat} ';' ;

beat                = sound {sound} ;

sound               = note | chord | '.' | '%' ;

chord               = '(' note {note} ')' ;

note_name           = 'C' | 'D' | 'E' | 'F' | 'G' | 'A' | 'B' ;

note                = note_name {('+' | '-' | '#')} ;

integer             = digit {digit} ;

identifier          = lower_case_letter {letter | digit} ;
```

9 附录二：MRL 语言实例

9.1 《王老先生有块地》

《王老先生有块地》的吉他演奏的五线谱如下图所示：



对应的 MRL 语言源代码如下：

```
BPM: 150;
INSTRUMENT: 0;  //Acoustic guitar(steel)
CUR_OCTAVE: 3;
VELOCITY: 1;
TEMPO: 4;

MUSIC oldMcdonald {
    = (E+E-) | E+ | E+ | B ;
    = (C+#E-) | C#+ | B | . ;
```

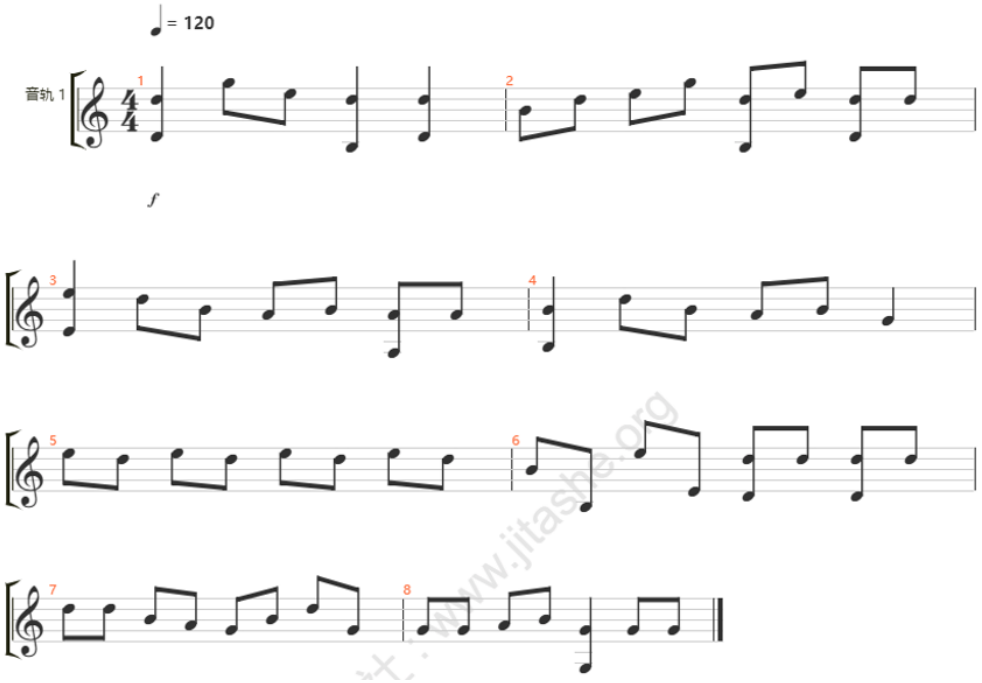
```
    = (G+#A-) | G#+ | (F+#B) | F#+ ;
    = E+ | B- | C# | D# ;

    = (E+E-) | E+ | (E+B) | B ;
    = (C+#E-) | C+# | (BB-) | B ;
    = (G+#A-) | G#+ | (F+#B) | F#+ ;
    = E+ | E- | B | C+# ;

    = E+ | E+ | E- | . ;
    = E+ | E+ | E- | . ;
    = E+ | E- | E+ | E- ;
    = E+ | E+ | B | C+# ;

    = E+ | E+ | A- | . ;
    = E+ | E+ | A- | . ;
    = E+ | A- | E+ | A- ;
    = E+ | E+ | B | C+# ;

    = (E+E-) | E+ | (E+B-) | B ;
    = (C+#E-) | C#+ | (BB-) | B ;
    = (G+#A-) | G#+ | (F+#BF#) | F+# ;
    = (E+E-) | . | . | . ;
}
```

9.2 《小兔子乖乖》

《小兔子乖乖》的吉他演奏的五线谱如下图所示：

对应的 MRL 语言源代码如下：

```
BPM: 120;
INSTRUMENT: 25;       //Acoustic guitar(steel)
CUR_OCTAVE: 3;
VELOCITY: 1;
TEMPO: 4;

MUSIC bunny {
    = (DD+) | G+E+ | (B-D+) | (DD+) ;
    = BD+ | E+G+ | (D+B-)E | (DD+)D+ ;
    = (EE+) | D+B | AB | (A-A)A ;
```




```
        = (B–B) | D+B | AB | G ;
        = E+D+ | E+D+ | E+D+ | E+D+ ;
        = BB– | E+E | (DD+)D+ | (DD+)D+ ;
        = D+D+ | BA | GB | D+G ;
        = GG | AB | (G–G) |GG ;
    }
```

9.3 《溜冰圆舞曲》

《溜冰圆舞曲》的吉他演奏的五线谱如下图所示：

对应的 MRL 语言源代码如下：

```
BPM: 192;
INSTRUMENT: 0;  //Acoustic grand piano
CUR_OCTAVE: 3;
VELOCITY: 1;    //strong
TEMPO: 3;

SEGMENT theme1{
    = (C+#A–) | E | E ;
    = (A–E+) | E | (F+#E) ;
    = (A–F+#) | D | D ;
    = A– | D | D ;
```

```
    = (A–D+) | F# | F# ;
    = (A–F+#) | F# | (G+#F#) ;
    = (G–#G+#) | E | E ;
    = G–# | E | E ;
    = (F–#B+) | B– | B– ;
    = (A–A+) | E | (EC+#) ;
    = (A–E+) | E | E ;
    = (B–D+) | E | (EC+#) ;
    = (A–C+#) | E | E ;
    = (B–B) | E | E ;
    = (A–A) | E | C# ;
    = A– | EA+ | F+#A+ ;
}

SEGMENT theme2{
    = (C#E+)A+ | (F+#E)A+ | (E+E)A+ ;
    = (C#F+#)A+ | (E+E)A+ | (F+#E)A+ ;
    = (E+A–) | (G+#E) | (G+#E) ;
    = (E+G–#) | (G+#E) | (G+#E) ;
    = (E+B–)B+ | (F+#E)B+ | (E+E)B+ ;
    = (F+#B–)B+ | (E+E)B+ | (F+#E)B+ ;
    = (E+A–) | (EA+) | (EA+) ;
    = (A–E+) | (EA+) | (EA+) ;
    = (EE+)A+ | (G+#A–)A+ | (E+C#)A+ ;
    = (A–E+)A+ | (G+C#)A+ | (EE+)A+ ;
    = (A–D+)A+ | (F+#D)A+ | (F#D+)A+ ;
    = (A–D+)A+ | (F+D)A+ | (F#D+)A+ ;
    = (A–C+#)A+ | (EE+)A+ | C+A+ ;
    = (B–D+)G+# | (EE+)G+# | D+G+# ;
}

MUSIC skatersWaltz {
    theme1;
    theme2;
    = (EA+) | . | (C#G+#) ;
    = (A+E)D | C#B– | A– ;

    theme1;
```

```
        theme2;
        = (ADBE) | . (G+#BD+) | (A+E+C+#) ;
        = (A-EA) | . | . ;
    }
```