HOLLABACH
Language Reference Manual

COMS W4115: Programming Languages and Translators

Craig Darmetko (cd2698)

July 2, 2014

1. Introduction

HOLLABACH is a language for composing MIDI based music. With HOLLABACH, composers can create compositions similar to writing programming code and utilize common programming tools and resources with their compositions. The language also includes support for loops and conditional logic for the often repetitive or algorithmic nature of musical compositions. HOLLABACH is designed to help composers create music efficiently, compose music collaboratively and abstract away the semantics of the MIDI specification.

2. Lexical Conventions

2.1 Comments

A comment is preceded by '/*' and ended with '*/'. These comments may span several lines but cannot be embedded in another comment.

2.2 Identifiers

Identifiers are a sequence of letters, digits, underscores and the '#' symbol. Identifiers cannot begin with a '#' symbol or a digit and cannot be a reserved keyword.

2.2.1 Reserved Keywords

The group of keywords that exist for defining language related function functionality are:

loop, if, else, bpm, m, measureLen, comp, null

These keywords cannot be used for identifiers.

2.3 Whitespace

A whitespace character is used to separate tokens and is otherwise ignored. Whitespace characters include the space, tab and newline characters.

3. Basic types/constants:

3.1 Integers:

Integer Constants are a sequence of digits separated by whitespace. Some uses for integers include representing conditional if statements, beats per minute declarations, measureLen declarations and loop iteration counts.

3.2 Note

A Note is a single contiguous musical sound. An example note is:

A#3/4

This defines a note of pitch A# in octave 3 with a length of a quarter note (1/4). The backslash denotes the beginning of the declaration of note length. Length is determined by the denominator of the fraction of a measure it lasts, such as 4 for quarter note, 2 for half note, 3 for triplets. Valid lengths include:

1 - whole note
2 - half note
3 - triplet
4 - quarter note
8 - eighth note
16 - sixteenth note.

The available pitches include A-G and an additional # or b character to represent sharp or flat notes. The octave section can contain a single digit between 1 and 4. The overall lexical pattern for identifying a note is

[‘A’-‘G’][‘#’|’b’]?[‘1’-‘4’][’/’][‘1’|’2’|’3’|’4’|’8’|’16’]

3.3 null

A null keyword is used in a measure to depict a beat in a measure that is a complete musical rest.

4. Complex types and Expressions

4.1 Chord

A chord is a group of notes that will be played simultaneously. An example chord:

C2/4+G3/4+A3/4+C3/4

This defines a chord of four notes that lasts for a quarter beat. Notes in the chord may end at different time, the goal is that all of the notes in a chord start at the same time. A '+' character separates notes and designates they are part of the same chord. A '+' may only precede a note if another note is following. For example:

C2/4+G3/4+ is not valid and
C2/4++G3/4 is not valid.

4.2 Measure

A Measure is a musical unit in a composition. It contains an array of notes,chords and nulls. Nulls represent a complete musical rest where nothing is playing. The position in the array indicates when the note or chord is played based on the length of the array. The array can be up to 32 units in length and it subdivides the measure into as many note start positions as there are array entries. For example,

m[A2/8, A2/8, A2/8, A2/8]

will play an eighth note every quarter beat for the measure. The array is declared using 'm'(short for measure) and the entries are surrounded by '[' and ']' and split using ','. There may not be a trailing ',' character before the ending ']' or a leading ',' character.

4.2.1 Declaring a Measure

Measures may also optionally be declared with an identifier for reuse in later lines in a composition. For example:

 m aEveryFour [A2/8, A2/8, A2/8, A2/8]

It can then be used in later lines with simply:

m aEveryFour

If a measure identifier is redefined the compiler will override the current definition with the new definition, but all previous uses of it will remain unchanged.

4.3 Composition

A composition combines measures into an overall structure and flow. It contains a list of sequential measures and optional loop statements. Compositions must have at least one measure or loop to be valid. Similar to the main method in Java or C, only one Composition section is allowed per file. This is what is converted to MIDI during compilation. Compositions are declared using the ‘comp’ tag.

```
comp{
    !BPM=140
    !measureLen=4
    m[G2/8, A2/8, B2/8, C2/8, D2/8, E2/8, F2/8, G3/8]
    loop 2 {
        m[G2/4+C2/4+G3/4,null,G2/4+C2/4+G3/4, null]
    }

}
```

The composition definition appears between the ‘{‘ and ‘}’ characters. Each measure, if statement or for loop is separated by a newline character and proceed sequentially. The beats-per-minute and measureLen are declared first and second in the composition and these together dictates the length of a single measure. The beats-per-minute and measureLen must always be declared and must be declared at the beginning of the composition definition.

4.4 Beats Per Minute

The Beats per Minute (BPM) declaration occurs at the beginning of a composition and determines the length of a musical beat in absolute time. BPM is declared with the tag ‘!BPM=’ and defined with a positive integer that follows. An example declaration is:

```
    !BPM=120
```

4.5 measureLen

MeasureLen is an attribute that determines how many beats occur in a measure. Dividing this value by the BPM gives you the absolute time length of a single measure. MeasureLen is declared with the tag ‘!measureLen=’ and defined with a positive integer that follows. An example declaration is:

```
    !measureLen=4
```

4.6 Loops

A loop statement allows a measure or set of measures to be repeated a given amount of times. A loop is declared with the keyword ‘loop’ and followed by an integer representing the number of repetitions. Following the integer, the loop contents are declared with a leading ‘{‘ and a trailing ‘}’. For example:

```
loop 2 {
        m[G2/4+C2/4+G3/4,null,G2/4+C2/4+G3/4, null]
}
```

A loop contents can include any combination of measures, if statements or embedded loops.

4.7 Conditional Statements

A composer can use conditional statements to change the behavior of certain loop iterations in a loop. The composer declares the conditional using an if statement, followed by the loop iteration number that the if statement should trigger. The iteration starts at 0 and increments until it is equal to the specified number. This means an if statement meant to trigger on the first iteration should use 0 as it's conditional. A composer can also trigger on the negation of a conditional by appending a '!' character to the front. This allows composers to define a sequence that is repeated on each loop except the given loop. The composer can also use an else statement after an if to trigger on the negation case of the if statement. The if body is declared after the conditional between brackets. The body can contain measures or additional loops.

5. Variable Scope

5.1 Identifier

The scope of a measure identifier in HOLLABACH is global to the composition once it is defined. Any references before it is defined in the composition will cause compiler errors.

5.2 Loop

A loop has an implicit variable, the loop iteration number. The scope of this variable is inside the loop (between the brackets) and is exclusively used in if statements. An if statement conditional always refers to the lowest loop. For example, if we have loop B embedded in loop A and an if statement in loop B, the conditional will refer to the iteration number of loop B.

6. Compilation and Output

To compile the program, HOLLABACH converts the Composition element to a series of notes by evaluating loops and conditionals. These notes are then translated into an intermediate CSV language and written out to a temporary file. From here, the CSV is read by the third party Java tool CSV2MIDI[1] to convert the data to a MIDI file. A sample CSV can be found at [2].

[1]- https://midilc.googlecode.com/svn/trunk/CSV2MIDI/CSV2MIDI.java
[2]- https://midilc.googlecode.com/svn/trunk/CSV2MIDI/ExampleMIDI.csv

7. Example

```
comp{
    !BPM=140
    !measureLen=4


    m[G2/8, A2/8, B2/8, C2/8, D2/8, E2/8, F2/8, G3/8]
    loop 2 {
        m GChord [G2/4+C2/4+G3/4,null,G2/4+C2/4+G3/4, null]
        m GChord
        m GChord
        if 1{
                /* plays on last loop */
                m[G2/4+C2/4+G3/4,null, null, null,G2/4+C2/4+G3/4, null, null, A3/8]
         }
         else{
                /* plays on all others */
                m GChord
         }
    }
    /* last note*/
    m [G2/1]
}
```