Rhythm

John Sizemore (Team Leader)
Cristopher Stauffer
Yuankai Huo
Lauren Stephanian

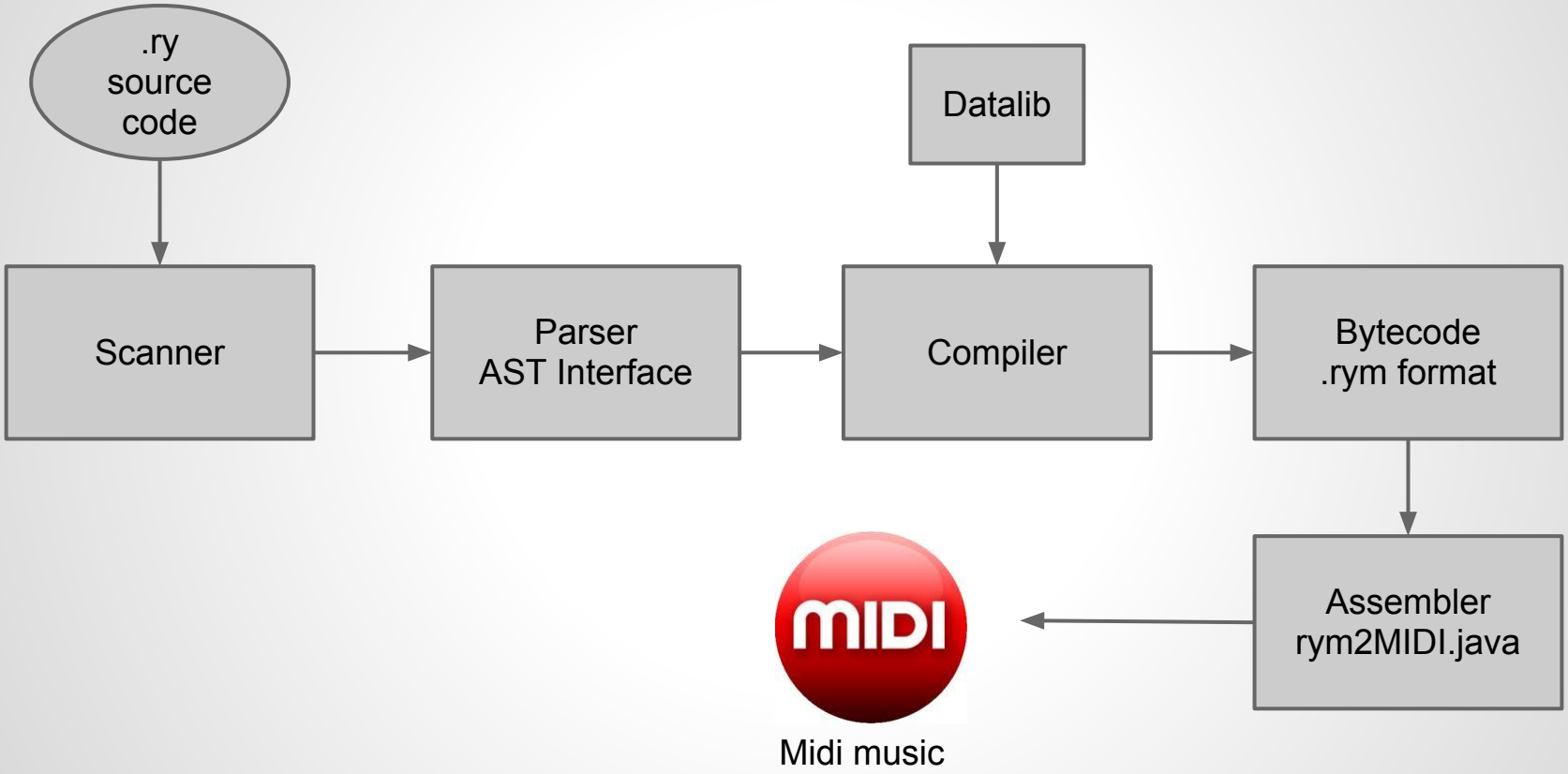
Introduction

- Rhythm is a music composition language
- Programmers create chronological tracks out of notes, rests, and chords
- Tracks can be played alone or with other tracks to create more complex music

Motivation

- Most music composition programs rely on visual or audio cues
- Furthermore, these programs often come with a substantial learning curve and require extensive knowledge about production and/or music theory
- Rhythm seeks to provide a simpler way to make music without requiring production experience
- Perfect marriage of music and programming

Project Architecture



Program Structure

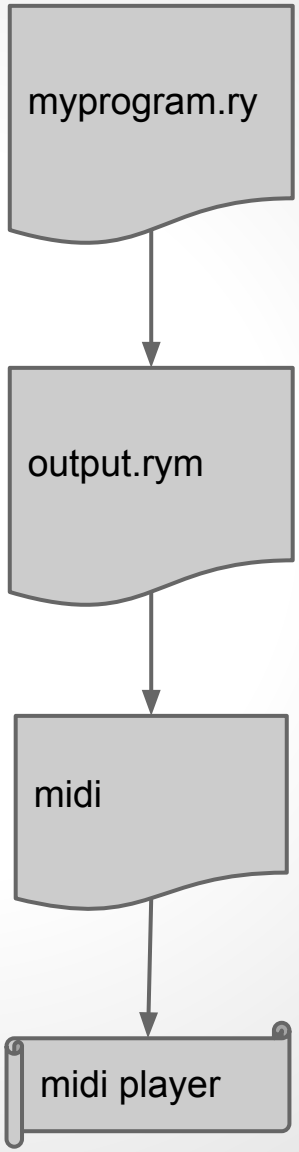
- Global Variable Definition
- Initialization Function Definition
- General Function Definition
- Track Function Definition

```
def s;                    /* Global Variable */


track_foo()
{
  c = [[A0.16,A1.16,A2.16],A3,A4.16,R.8,A2];
  return c;               /* Local Variable */
}

track_foo2()
{
  return s;
}

init()
{
  s = [A5,B3,R.1,D7];     /* OK */
  c = c >> 2;                  /* Error! */
}
```

Program Output

Track: foo
1 0
1 12
1 24
2 36
3 36
4 36
5 36
6 48
9 24
10 24
11 24
12 24

Track: foo2
1 60
2 60
3 60
4 60
5 38
6 38
7 38
8 38



General Language Properties

- Imperative - Function Based Language
- Static Variable Scoping Rules
  - Global variables are defined at top of program with “def” keyword.
  - Local variables are defined as function parameters or as expressions in the function body.
  - Variables must be defined before they are used
- Static Typing - Although variable typing is inferred instead of explicitly defined. No “note” or “chord” keywords.
- No standard “write” procedure - compiling a track accomplished via `return` statements from `track` functions. Better design for modularity and for separating tracks.

Keywords

```
if
else
loop
while
```

```
true
false
return
```

Special Function Names

```
init()
track_*()
```

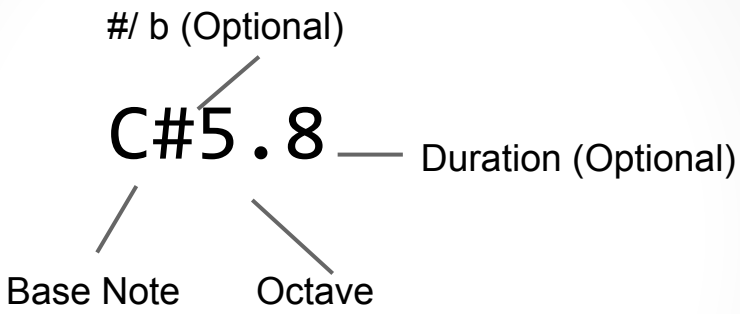
Variable and Function Definition

- Variables can be global or local
  - Globals defined using the ‘def’ keyword (e.g. `def x`)
  - Locals defined by simple assignment: (e.g. `c = A4`)
  - Definition and assignment must be a separate operation for global variables
- Function definition is of the form:

```
function_name(param_1,...,param_n)
{
    def x;
    … statements...
    return z;
 }
```

Primitive Types

- Ids, Integers
- Notes



- Rests

- Array
  - Tracks e.g. `[C5,[A1,A2,A3],G#6.8]`
  - Chords e.g. `[A1,A2,A3]`

Expressions and Statements

- Unary Expressions
  - Notes, Rests, Literals
- Assignment
  - `note = C#5`
- Array access
  - `myArray[5]`
- Binary Operation
  - x OP y
- Statements
  - end in semicolon

Operators

- **Modification Operators**
  - ‘+’ ‘-’ ‘++’ ‘--’ ‘*’ ‘/’ ‘<<’ ‘>>’
- **Combinational Operators**
  - expression -> expression
  - expression :: expression
- **Equality Operators**
  - expression == expression
  - expression != expression
- **Assignment Operators**
  - lvalue = expression
  - lvalue += expression
  - lvalue -= expression
  - lvalue *= expression
  - lvalue /= expression
  - lvalue >>= expression
  - lvalue <<= expression
  - lvalue ::= expression

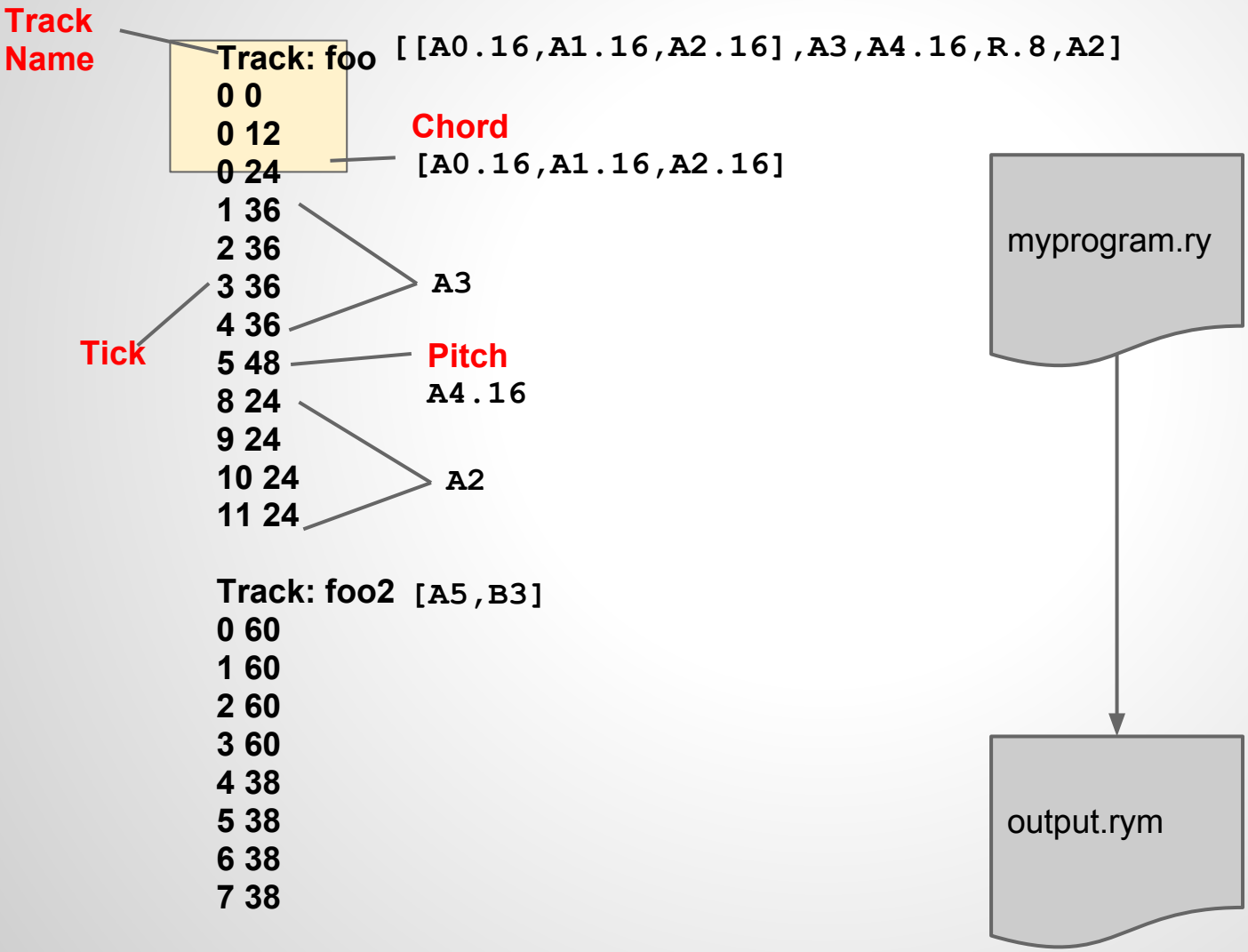
Operators II

- **+**
  - Arithmetic: `1 + 1 = 2`
  - Pitch changes: `C4 + 1 = C#4`
  - Mixing: `[A4, B4] + [C4, D4] = [[A4, C4], [B4, D4]]`
- **-**
  - Minus: Same principles apply with arithmetic and pitch changes
  - Cannot “de-mix”. Mixing operation constructive only
- **++/--**
  - Shorthand for increasing/decreasing value/pitch: `C4++ = C#4`
- **>>/<<**
  - Octave Shifting: `C4 >> 1 = C5`
- ***
  - Increase note duration: `C4.4 * 2 = C4.2`
  - Seems counterintuitive, but notes can be represented as either whole, half, quarter, eighth, sixteenth notes
  - C4.4 is a quarter note : `C4.4 * 2` changes it to a half note (`C4.2`)
- **/**
  - Decrease note duration

Operators III

- **::**
  - Concatenation: `[A4, B4] :: C4 => [A4, B4, C4]`
  - Useful for sequentially ordering tracks
- **->**
  - Stretch: `R4.1 -> 2 => [R4.1, R4.1]`
  - Useful for padding or making loops
- **==**
  - Equality Check
  - `A4 == B4 = false`
  - `[A4, B4, C4] == [A4, B4, C4] = true`
- **!=, >, >=, <, <=**
  - Inequality Check
  - `A4 != B4 = true` `A4 < B4 = true`
  - `[A4, B4, C4] != [A4, B4, C4] = false`
- **=**
  - Assignment: `c = [A4, B4, C4];`
- **+=, -=, *=, /=, ::=, >>=, <<=**
  - Performs operation and assigns result to the lvalue on the left
  - `c ::= D4 = [A4, B4, C4, D4]`

Rym File Format



Generate Midi

output.rym

Track 1
0 0
0 12
0 24
1 36
2 36
3 36 note 1
4 36
5 48 note 2
8 24
9 24
10 24
11 24 note 3
12 24
13 24
14 24

Track 2
0 60
1 60
2 60
3 60
4 38
5 38
6 38
7 38

step1: Generate Tick Table

pitch	ticks
0	[0]
12	[0]
24	[0, 8, 9, 10, 11, 12, 13, 14]
36	[1, 2, 3, 4]
48	[5]

step2: Generate Onset Duration

pitch	onset	duration
0	0	1
12	0	1
24	0	1
	8	7
36	1	4
48	5	1

step3: Send Message To Track

```
track[1].addmessage(0, 0, 1)
track[1].addmessage(12, 0, 1)
track[1].addmessage(24, 0, 1)
track[1].addmessage(36, 1, 3)
track[1].addmessage(36, 4, 1)
track[1].addmessage(48, 5, 1)
track[1].addmessage(24, 8, 4)
track[1].addmessage(24, 12, 3)
```

output.midi

Complete Program

Row Row Row Your Boat

```
getBaseNotes() {
    def row;
    def rowbase;
    rowbase = [[C5,E5,G5], [C5,E5,G5], [C5,
E5,G5],D5.8, E5.8, E5.8, D5.8, E5.8, F5.8,
G5.2, C6, G5, E5, C5, G5.8, F5.8, E5.8, D5.
8, [C5,E5,G5]];

   row = rowbase->3;
   return row;
}

track_1() {
    return getBaseNotes();
}

track_2() {
    return R.1->4 :: getBaseNotes() << 2;
}

track_3() {
    return R.1->2 :: getBaseNotes() << 1;
}
```

Complete Program

```
track_1() {
  c = [[C5.1,C6.1,C4.1,C3.1,C2.1]];   /* C octaves */
  e = c + 4;                          /* E octaves */
  g = c + 7;                          /* G octaves */
  count = 0;
  song = [];
  while (count < 12) {
         song = song :: (c+e+g) :: R.1->2 :: (c+1 +
     e+1 + g+1) :: R.1->16;
         c++; e++; g++; count++;
     }
  return song->3;
  }
```

Complete Program

Shepard Tones

- Audio Illusion
- Repeated sequence of notes that sound like they are always rising in pitch
- Works better with certain sounds than others
- Simple waveforms (e.g. sinusoid) work best

```
track_1() {
  c = [[C5.1,C6.1,C4.1,C3.1,C2.1]];   /* C octaves */
  e = c + 4;                          /* E octaves */
  g = c + 7;                          /* G octaves */
  count = 0;
  song = [];
  while (count < 12) {
         song = song :: (c+e+g) :: R.1->2 :: (c+1 +
     e+1 + g+1) :: R.1->16;
         c++; e++; g++; count++;
     }
  return song->3;
  }
```

Complete Program

An example of a pop music
1. popular
2. released in 2012

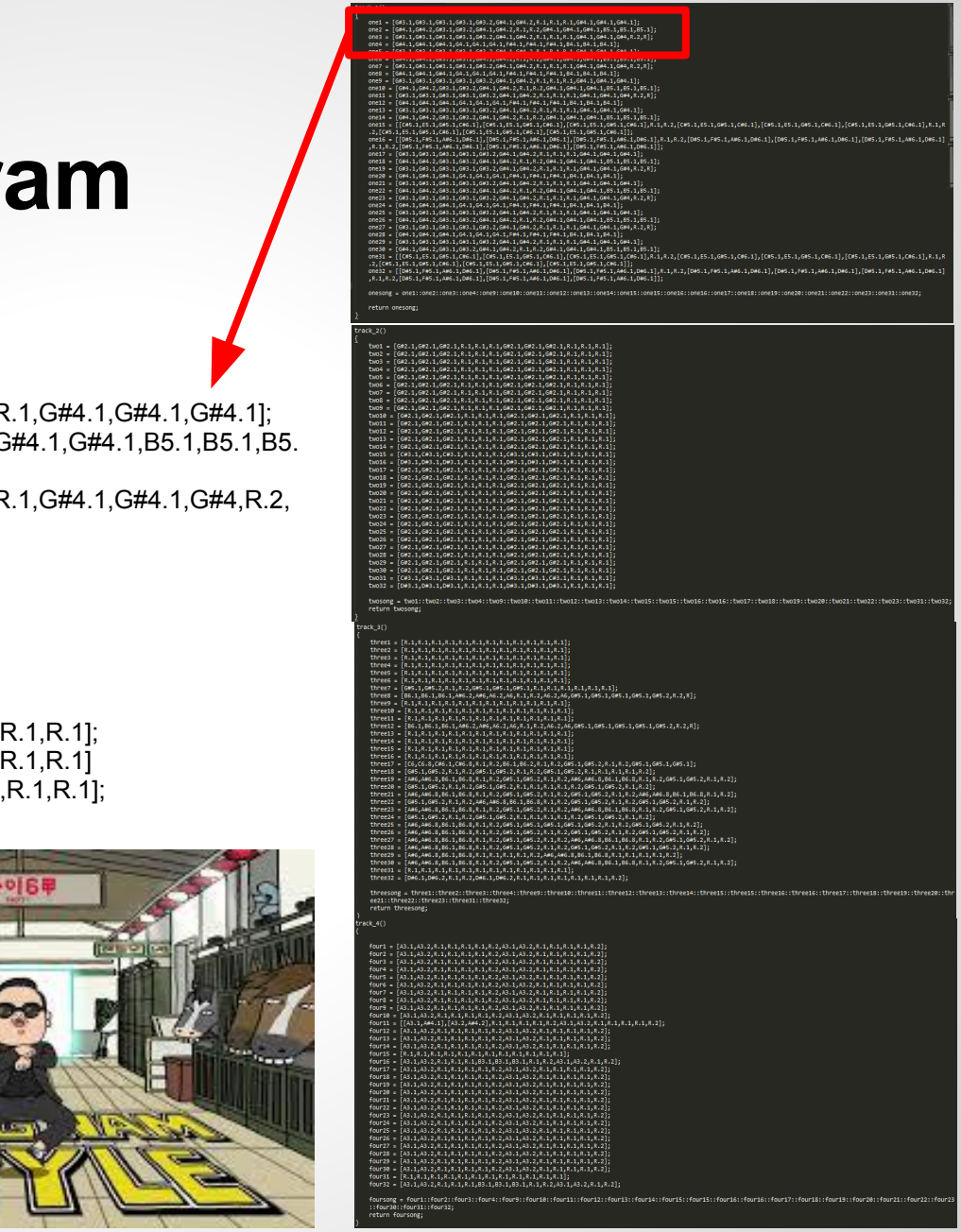
Can you recognize this music?

More important
Rhythm supports multi-tracks !

Complete Program

multitracks example

```
track_1(){
one1 = [G#3.1,G#3.1,G#3.1,G#3.1,G#3.2,G#4.1,G#4.2,R.1,R.1,R.1,G#4.1,G#4.1,G#4.1];
one2 = [G#4.1,G#4.2,G#3.1,G#3.2,G#4.1,G#4.2,R.1,R.2,G#4.1,G#4.1,G#4.1,B5.1,B5.1,B5.
1];
one3 = [G#3.1,G#3.1,G#3.1,G#3.1,G#3.2,G#4.1,G#4.2,R.1,R.1,R.1,G#4.1,G#4.1,G#4,R.2,
R];
…

onesong = one1::one2::one3 …
return onesone}

track_2(){
two1 =[G#2.1,G#2.1,G#2.1,R.1,R.1,R.1,G#2.1,G#2.1,G#2.1,R.1,R.1,R.1];
two2 =[G#2.1,G#2.1,G#2.1,R.1,R.1,R.1,G#2.1,G#2.1,G#2.1,R.1,R.1,R.1]
two3 = [G#2.1,G#2.1,G#2.1,R.1,R.1,R.1,G#2.1,G#2.1,G#2.1,R.1,R.1,R.1];
…


twosong = two1::two2::two3 …
return onesone}

track_3(){
…}

track_4(){
…}
```



Conclusions

- Language Learnings
  - Initially difficult to think of language as anything other than a configuration
  - .rym data can be easily changed: fairly straightforward
- Project Learnings
  - An early start is extremely beneficial
  - Weekly meetings and maintaining communication are very important
  - Modular division of tasks critical
  - Now, we not only know how to drive a car (use c,java ...) but also know how to build one!