MIDILC

Fredric Lowenthal, Ye Liu, Akiva Bamberger, Ben Mann

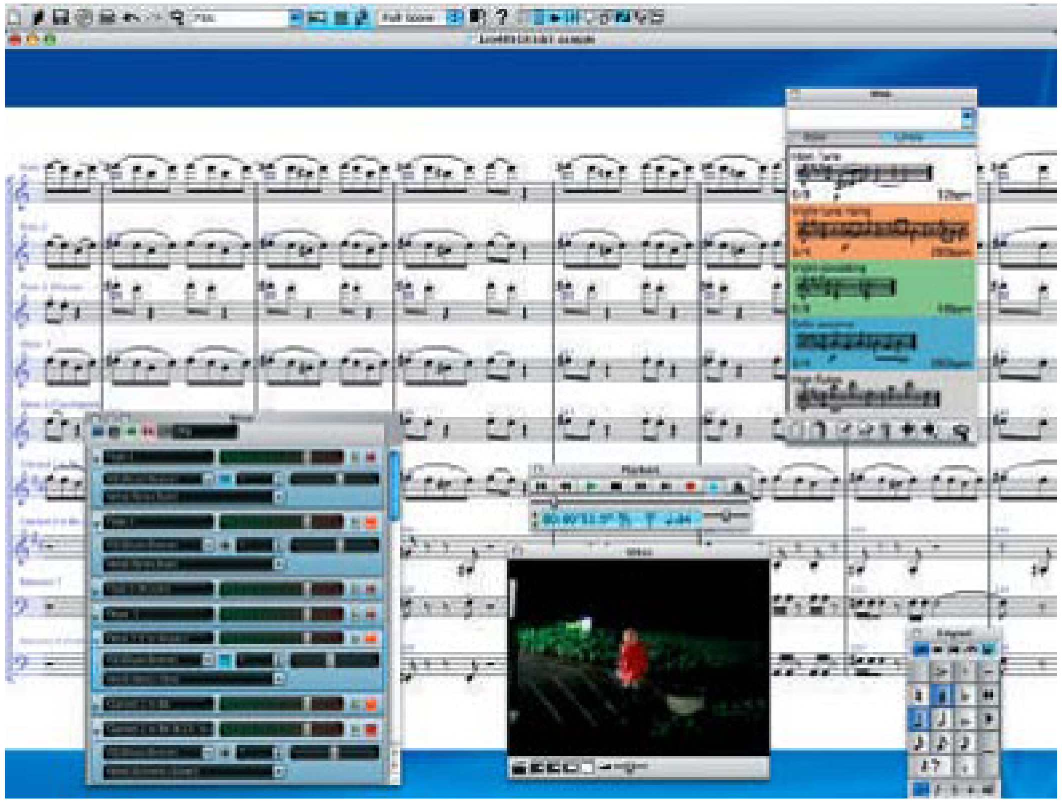
Outline

- Overview
- Tutorial and demo
- Implementation
- Lessons

Overview

Akiva

- Music programs like Sibelius require a lot of point and click action.
- Not nerd friendly!



MIDILC

Akiva

- Language is structured to help nerds build music quickly.
- Structure of the language is broken into several types:
  - `Void`
  - `Number` - a 32 bit signed integer which can be used for math and logic
  - `Note` - a musical atom consisting of two `Number`s, pitch and duration, and represented by one of several Note literals matching regex `[A-G R][b#][0-9][w h e s q]`
  - `Chord` - a collection of `Note`s with same start time + duration (represented as list of `Number`s)
  - `Sequence` - a collection of `Chord`s (represented as list of list of `Number`s)

More about MIDILC

Akiva

- Dynamically typed language, with type declarations necessary for variable declarations and optional for functional declarations and parameters
- Statically scoped with applicative order
- Fun for the whole family!

Say hello to your new instrument!

What's included?
Akiva

- Built in functions for several important features, such as `play()`, `set_instrument()`, `set_tempo()`, `new_chord()`, and `new_sequence()`
- Bytecode + CSV as Intermediate Representation

Beethoven says "Writing symphonies in MIDILC is fun and makes me giggle. Tee hee!"

MIDILC Basics

Fred

- All MIDILC programs must have a `main()` function that includes a `play()` statement, in order to generate an output.
- Declarations must come before any other statements; they can't be intermingled.
- A sequence must be passed into the `play()` function
- `set_instrument()` and `set_tempo()` can be used to set the instrument via a string with the instrument's name, and a number with the tempo in BPM, respectively. If they are both used, they must be called in that order, before the `play()` function

MIDILC Basics
Fred

- A simple program:

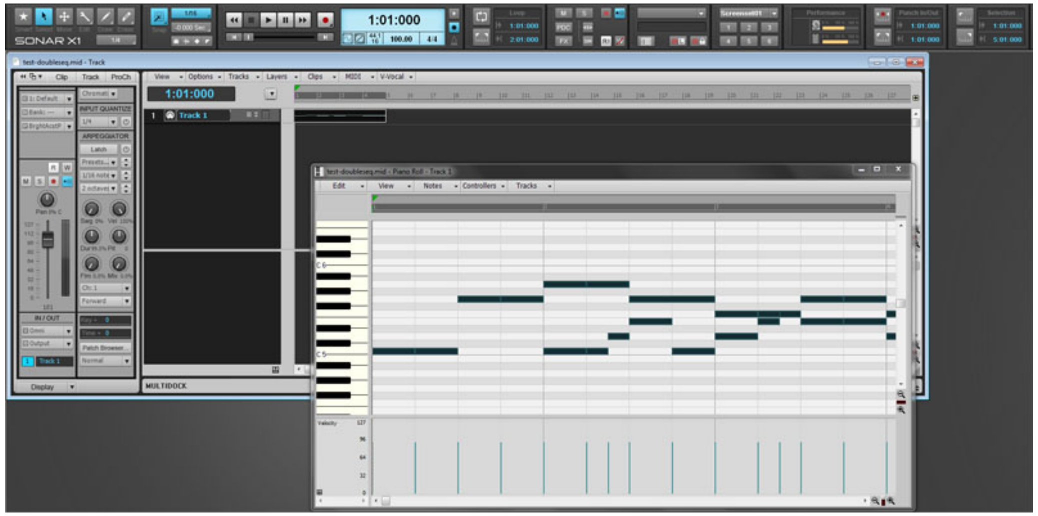
```
main() {
   Chord cMajor;
   Note root;
   Sequence seq;

   root = C4;
   cMajor = new_chord(root, root .+ 4, root .+ 7);
   seq = new_sequence();
   seq = seq + cMajor;
   play(seq);
 }
```

MIDILC Basics

Fred

- The sample program creates a `Note`, `Chord`, and `Sequence` object, and then plays the sequence, composed of one chord (the C major chord).
- As this example shows, music can be composed using simple mathematical operations (in this case, numerically instantiating a major chord from a root); the `.+` operator indicates an addition operation that uses the `pitch` property.



Tutorial:
Twinkle, Twinkle

Declaring Variables

Ye

```
main(){
  Chord ch1;
  Chord ch2;
  Chord ch3;
```

Declare all variables

Declaring Variables

Ye

```
main(){
  Chord ch1;
  Chord ch2;
  Chord ch3;
  Sequence s;
  Number i;
  Number r1;
  Number r2;
```

Declare all variables

Initializing Variables

Ye

```
main(){
  Chord ch1;
  Chord ch2;
  Chord ch3;
  Sequence s;
  Number i;
  Number r1;
  Number r2;
  ch1 = new_chord(C,E,G);
  ch2 = new_chord(C,F,A);
  ch3 = new_chord(G3s,B3s,D4s,F4s);

  s = new_sequence();
```

Initialize Chord and Sequence

Building a Sequence

Ye

```
main(){
  Chord ch1;
  Chord ch2;
  Chord ch3;
  Sequence s;
  Number i;
  Number r1;
  Number r2;
  ch1 = new_chord(C,E,G);
  ch2 = new_chord(C,F,A);
  ch3 = new_chord(G3s,B3s,D4s,F4s);
  s = new_sequence();
  s = s + C + C;
  s = s + ch1 + ch1 + ch2 + ch2 + ch1;
  s = s + arpeggiate(ch3) + F + F;
  s = s + E + E + D + D + C;
```

Add Notes, Chords, and Sequence returned by arpeggiate()

Tempo and Play

Ye

```
main(){
  Chord ch1;
  Chord ch2;
  Chord ch3;
  Sequence s;
  Number i;
  Number r1;
  Number r2;
  ch1 = new_chord(C,E,G);
  ch2 = new_chord(C,F,A);
  ch3 = new_chord(G3s,B3s,D4s,F4s);
  s = new_sequence();
  s = s + C + C;
  s = s + ch1 + ch1 + ch2 + ch2 + ch1;
  s = s + arpeggiate(ch3) + F + F;
  s = s + E + E + D + D + C;
  set_tempo(125);
  play(s);
}
```

Set tempo and play the song as a CSV

The `arpeggiate()` function

Ye

```
Sequence arpeggiate(Chord chord) {
  Number n;
  Number i;
  Sequence s;
  s = new_sequence();
  n = chord.length;
  for(i = 0; i < n; i=i+1) {
    s = s + chord[i];
  }
  return s;
}
```

function name

variable declarations

for loop
subscripting for Chord

return a Sequence

Bytecode

Ye

```
0 global variables
0 Jsr 36
1 Hlt
2 Ent 3
3 Jsr -3
4 Sfp 3
5 Drp
6 Lfp -2
7 Mem length
8 Sfp 1
9 Drp
10 Num 0
11 Sfp 2
12 Drp
13 Sjp (7,15,0)
14 Bra 13
15 Lfp 3
16 Lfp 2
17 Lfp -2
```

... etc

CSV output

Ye

```
Tempo,125
0,4,60
4,4,60
8,4,60
8,4,64
8,4,67
12,4,60
12,4,64
12,4,67
16,4,60
16,4,65
16,4,69
20,4,60
20,4,65
20,4,69
24,4,60
```

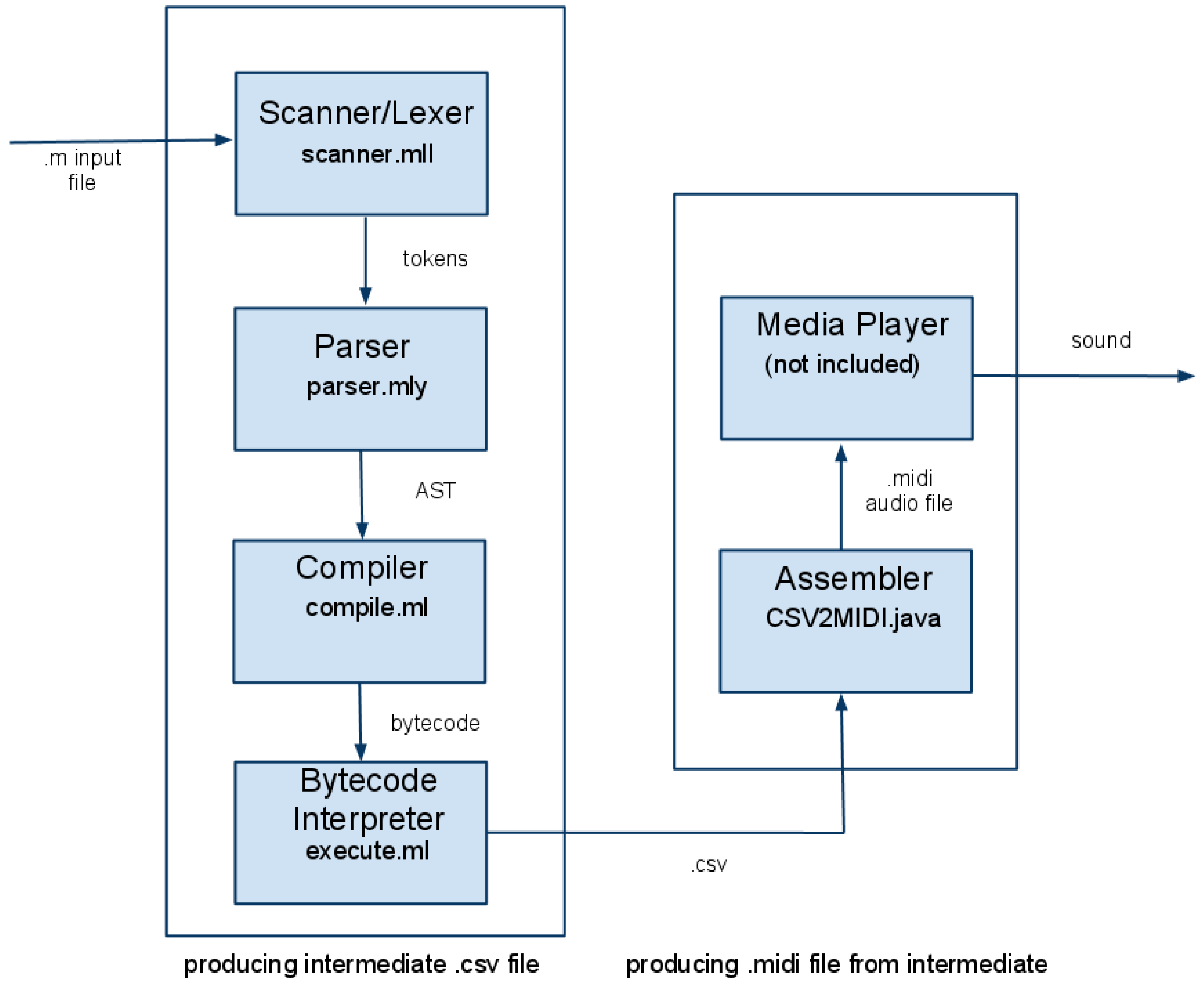
... etc

```
24,4,64
24,4,67
28,1,55
29,1,59
30,1,62
31,1,65
32,4,65
36,4,65
40,4,64
44,4,64
48,4,62
52,4,62
56,4,60
```

CSV file
(tick, duration, pitch)

Implementation

Ben



Compilation

Ben

- Turns AST into bytecode
- Special features
  - `Note` literals (e.g., `A`, `A#6h`)
  - Built in functions
    - Chord constructor varargs
  - `break` and `continue`

Execution

Ben

- Turns bytecode into CSV
- Stack holds bytecode objects
- Global and local variables also bytecode objects
- Assignment replaces the data in the lvalue with the rvalue
- Special features:
    - Subscripting and direct selection
    - Casting

Lessons

- Akiva:
  - Understand and complement teammates' strengths
  - Build and test
- Fred:
  - Good source control and tools save time
  - Work as a group, not a set of components
- Ye:
  - Testing is your friend
- Ben
  - Investing time in understanding
    - No manual? RTFM → RTFC
  - Command line