Lullabyte

Stanley Chang Louis Croce Nathan Hayes-Roth
Andrew Langdon Ben Nappier Peter Xu

Overview

- Goals
  - Generate MIDI music files
  - C-like syntax
  - Allow algorithmic or direct composition
- Building blocks
  - Construct sounds from ints, doubles, pitches
  - Construct tracks from series of sounds
  - Layer tracks into a song

Program Structure

- main function

  `void main() {...}`

- global variable declarations

  `int global_i; sound[] sounds;`

- function definitions

  type function (*types args*) {...}

Functions

```
type function(){
        // local variable declarations
        // global variable assignments
        // control flow
        // other function calls
        // optional return value
}
```

Arrays

- Dynamic. Length is not fixed
- Assigning element beyond length pads intermediate elements with default type values (0, 0.0, false, C0, |C0|:0.0:0, etc.)
- Accessing element beyond length throws IOB error
- Reason: Make it easier for developer to not worry about checking array lengths since musical tracks change a lot throughout development

Types

- int, double, boolean
- Pitch: C0, A1, Bb4, G9, …
- Sound: “<pitch(es)> : <double> : <int>”
  |C0|:0.25:100 |C5, E4, G3|:0.25:70

Statements & Control Flow

- if, while, for, return
- loop (<var> : <array>) {

      //body

  }
- Reference to <var> in body is treated as “array[i]”
- Easy way to loop through sound arrays and make modifications with cleaner code

Built-in Functions

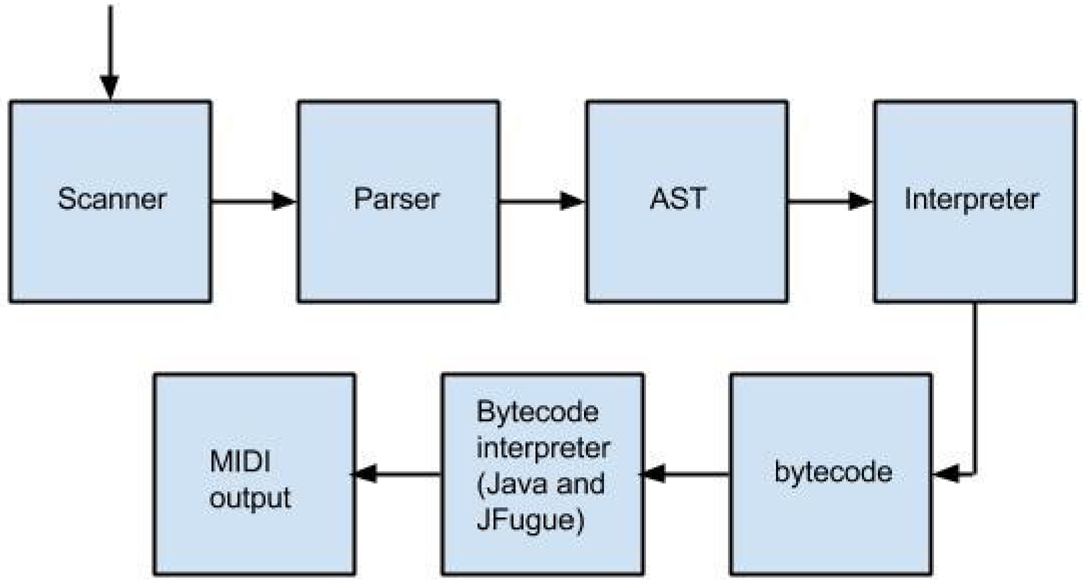
- setPitches(<sound>, <pitch>)
- setDuration(<sound>, <double>)
- setAmplitude(<sound>, <int>)
- getPitches(<sound>)
- getDuration(<sound>)
- getAmplitude(<sound>)

- length(<array>)
- randomInt(<int>)
- randomDouble(<double>)
- bpm(<int>)
- write()
- play()
- print(<expr>)
- mixDown(<sound[]>, <int>)

mixDown(sounds, track)

- most important built-in function
- writes array of sounds to midi
- programmer specifies track number
- can be called multiple times
- sounds appended to specified track

Architectural Design



Front End

- scanner.mll
- parser.mly
- ast.ml

Back End

- interpreter.ml
  - rules of our compiler
- typechecking
  - variable type is stored on value declaration
  - function type is stored in module

Conversion to Midi

- Bytecode Conversion
  - catches .llb Failures
  - bpm, write, play, or both
  - tracks
- BytecodeTranslator.java
- JFugue

```
120 p
0[[Bb5]:0.5:100,[C6]:0.5:100,
[C6]:0.5:100, [Bb5]:0.5:100]
1[[Bb4, D4, G3]:1.:100,[E4, G3,
C3]:1.:100]
0[[Bb5]:0.5:100]
```

```
“V1 [72]/0.5a100+[76]/0.25a100+[76]/0.  5a100”...
```

Testing

- testing suite
    - *.llb and *.out
- type check test
    - breaking the compiler

Lessons Learned

- MicroC Slides
- Version Control
- Testing
- Strength in Numbers
- Communication
- Accountability

Hey Jude Demo

Thank You

Any Questions?